Discernimento entre Códigos-fonte Gerados por Pessoas e por Inteligência Artificial: Estudo com Programação em Java

Túlio José Prestrelo Miranda

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, PB, 2024

Túlio José Prestrelo Miranda

Discernimento entre Códigos-fonte Gerados por Pessoas e por Inteligência Artificial: Estudo com Programação em Java

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Carlo Gabriel Porto Bellini

Outubro de 2024

Catalogação na publicação
Seção de Catalogação e Classificação

M672d Miranda, Tulio Jose Prestrelo.
Discernimento entre códigos-fonte gerados por pessoas e por inteligência artificial: estudo com programação em Java / Tulio Jose Prestrelo Miranda. - João Pessoa, 2024.
82 f. : il.

Orientação: Carlo Gabriel Porto Bellini.
TCC (Graduação) - UFPB/CI.

1. Teste de turing. 2. Código fonte. 3. Critérios heurísticos. 4. Java. 5. Inteligência artificial. I. Bellini, Carlo Gabriel Porto. II. Título.

UFPB/CI CDU 004.8

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado ***O poder das Inteligências Artificiais na Geração de Códigos-fonte*** de autoria de Túlio José Prestrelo Miranda, aprovada pela banca examinadora constituída pelos seguintes professores:

_____

Prof. Dr. Carlo Gabriel Porto Bellini
Universidade Federal da Paraíba

_____

Profa. Dra. Thaís Gaudencio do Rego
Universidade Federal da Paraíba

_____

Prof. Dr. Yuri de Almeida Malheiros Barbosa
Universidade Federal da Paraíba

João Pessoa, 30 de Outubro de 2024

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

"Só se pode alcançar um grande êxito quando nos mantemos fiéis a nós mesmos."
Friedrich Nietzsche

AGRADECIMENTOS

Ao meu pai, Julio Miranda do Nascimento Neto, à minha mãe, Kelly Christine Nascimento Prestrelo Miranda, e ao meu irmão, Gabriel Akylis Prestrelo Miranda por estarem ao meu lado durante toda a minha vida, me apoiando com muita dedicação e amor em minha jornada até este momento. Certamente foi graças a todo o esforço por parte deles que pude crescer, me tornar a pessoa que sou hoje e conquistar todas as vitórias que tive até então.

Aos amores de minha vida, minha esposa Beatriz Nayara Passos Rocha, que esteve e estará junto a mim durante todas as dificuldades, que é uma grande fonte de apoio e ajuda para que eu pudesse me manter sempre firme e continuar a caminhar para meu desejado futuro e ao meu filho Leonardo Miranda Passos Rocha Prestrelo que em um momento tão difícil pelo qual eu passei, foi minha principal fonte de motivação para seguir em frente.

Um agradecimento especial a duas pessoas que não se encontram mais entre nós, meu avô Marcos Prestrelo Marinho que é até hoje uma grande fonte de inspiração para mim, do que eu almejo me tornar para minha família, um pilar onde todos possam encontrar felicidade e conforto, e minha tia Janaína Idalice Nascimento Prestrelo Marinho que sempre levou a festa para onde quer que nós estivéssemos, trazendo consigo a felicidade e alegria.

Aos professores, que cumprem um papel tão importante na formação de todos os que passam por essa instituição de ensino, ajudando a construir um futuro melhor para nosso país, um aluno de cada vez. Em especial, ao professor Carlo Bellini, que confiou em minha capacidade o bastante para aceitar orientar esse final de uma fase muito importante da minha vida e que tem me incentivado de várias maneiras a crescer e continuar meu esforço de crescimento.

RESUMO

A revolução tecnológica impulsionada pela popularização das Inteligências Artificiais (IA) tem suscitado a necessidade premente de organizações manterem equipes de profissionais altamente qualificados em Tecnologia da Informação (TI), em áreas como a de desenvolvimento de software, para discernir entre criações humanas e produções geradas por máquinas. Esta monografia aborda o desenvolvimento de códigos fonte em Java gerados a partir das inteligências artificiais ChatGPT e Bard e submetendo a 10 alunos do CI-UFPB a identificarem se o código foi criado por uma máquina ou por um programador humano. O objetivo principal é estudar e entender as métricas e critérios heurísticos utilizados para a identificação de origem dos códigos utilizando o teste de Turing para extrair dados relevantes. O teste de Turing é um método que visa determinar se uma máquina pode exibir um comportamento inteligente indistinguível do comportamento humano. Neste contexto, o teste de Turing é adaptado para avaliar se o código gerado pela inteligência artificial é convincente o suficiente para ser confundido com o código humano pelos alunos. Para alcançar esse objetivo, a pesquisa se concentra em cinco critérios heurísticos para análise de código, incluindo a estrutura, a semântica e as características sintáticas. Esses critérios heurísticos são essenciais para o entendimento de geração de códigos fonte conforme os resultados, visto que 100% dos participantes afirmaram que os códigos foram úteis para a identificação da origem do código.

Palavras-Chave: <Teste de Turing>, <Código Fonte>, <Critérios Heurísticos>, <Java>, <Inteligência Artificial>.

ABSTRACT

The technological revolution driven by the popularization of Artificial Intelligence (AI) has raised the pressing need for organizations to maintain teams of highly qualified professionals in Information Technology (IT), in areas such as software development, to discern between human creations and productions generated by machines. This monograph addresses the development of source code in Java generated from ChatGPT and Bard artificial intelligence and asks 10 CI-UFPB students to identify whether the code was created by a machine or a human programmer. The main objective is to study and understand the metrics and heuristic criteria used to identify the origin of codes using the Turing test to extract relevant data. The Turing test is a method that aims to determine whether a machine can exhibit intelligent behavior indistinguishable from human behavior. In this context, the Turing test is adapted to assess whether the code generated by artificial intelligence is convincing enough to be confused with human code by students. To achieve this goal, the research focuses on five heuristic criteria for code analysis, including structure, semantics, and syntactic characteristics. These heuristic criteria are essential for understanding the generation of source codes according to the results, as 100% of participants stated that the codes were useful for identifying the origin of the code.

Key-words: <Turing Test>, <Source Code>, <Heuristic Criteria>, <Java>, <Artificial Intelligence>.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

DL - Deep Learning - Aprendizado profundo

IA - Inteligência Artificial

LLMs - Large Language Models - Grandes modelos de linguagem

LOC - Lines of Code - Códigos de linha

ML - Machine Learning - Aprendizado de Máquina

NLP - Natural Language Processing - Processamento Natural de Linguagem

TI - Tecnologia de Informação

Sumário

1. INTRODUÇÃO

A revolução tecnológica tem sido impulsionada pela popularização das Inteligências Artificiais (IA). Este fato tem suscitado a necessidade premente de organizações manterem equipes de profissionais altamente qualificados em Tecnologia da Informação (TI), em áreas como a de desenvolvimento de software, para discernir entre criações humanas e produções geradas por máquinas.

Critérios Heurísticos como: formatação [2], comentários [2], átomos de confusão [4], nome de identificadores [5] e número de linhas [35], são alguns dos critérios estudados em diversas situações. Uma situação, por exemplo, seria a de tentar entender as métricas empregadas para identificar a origem dos códigos gerados, ou seja, se o código gerado foi por um ser humano ou uma inteligência artificial.

Dentre todos os tópicos de pesquisa, um dos que mais se repete está relacionado ao estudo da legibilidade do código, definida como uma medida de quão facilmente os desenvolvedores podem ler e entender o código-fonte [9][10], e a legibilidade de um código pode, sim, impactar na percepção de sua origem, o que pode levar alguém a pensar que um código mais bem estruturado e fácil de ler veio de um ser humano, enquanto um código de difícil leitura seria associado a uma máquina [50]. Entender por que os profissionais da área de TI dedicam mais tempo nas horas de trabalho tem sido promissor, de forma que com esse conhecimento é mais fácil manter a constante manutenção e evolução do software [3], e isso resulta em melhores resultados na entrega de projetos e correção de erros.

O objetivo principal desta pesquisa consiste em aprofundar o estudo e a compreensão das métricas e critérios heurísticos empregados para identificar a origem dos códigos gerados por meio do teste de Turing adaptado.

O teste de Turing, originalmente concebido para determinar a capacidade de uma máquina em exibir um comportamento inteligente indistinguível do comportamento humano [1], é adaptado, neste contexto, para avaliar a capacidade das IA em produzir códigos que possam ser confundidos com criações humanas pelos avaliadores.

Para atingir esse objetivo, a pesquisa se concentra em uma gama de critérios heurísticos para análise de código. Essas ferramentas analíticas desempenham um papel crucial no entendimento da geração de código-fonte por IA e na busca por respostas para os desafios que essa revolução tecnológica impõe. Neste contexto, este trabalho visa contribuir

para a compreensão desse fenômeno e suas implicações na área de Tecnologia da Informação e Inteligência Artificial.

1.1. Definição do Problema

O problema central tratado neste documento gira em torno da capacidade das Inteligências Artificiais (IA), representadas pelas ferramentas ChatGPT e Bard, em gerar código-fonte em Java, que seja tão convincente e próximo da qualidade humana que possa confundir os avaliadores humanos.

Essa problemática é de relevância significativa no contexto atual, pois a automação da criação de código tem o potencial de impactar profundamente a indústria de TI, e a habilidade de distinguir entre código gerado por IA e código humano é fundamental para garantir a qualidade e segurança das aplicações de software. Portanto, este trabalho busca abordar essas questões, contribuindo para o entendimento desse desafio emergente na área de Tecnologia da Informação e Inteligência Artificial.

1.2. Objetivo Geral

O objetivo geral deste trabalho é investigar e analisar as métricas, critérios heurísticos e técnicas de avaliação de código-fonte em Java gerado por Inteligências Artificiais, especificamente ChatGPT e Bard, para determinar em que medida esses códigos podem ser confundidos com criações humanas. O trabalho estará concluído quando houver uma compreensão de quais métricas a IA produziu em códigos Java que se assemelharam tanto à qualidade e estilo de código humano que se tornem indistinguíveis para avaliadores humanos. Isso incluirá a identificação das métricas mais relevantes para essa avaliação, bem como a adaptação do teste de Turing como uma ferramenta eficaz para essa análise.

1.3. Objetivos Específicos

A partir do objetivo geral descrito, este trabalho possui os seguintes objetivos específicos:

- Analisar as características estruturais: identificar e analisar as características estruturais dos códigos-fonte em Java gerados pelas Inteligências Artificiais, comparando-as com os códigos escritos por humanos, utilizando métricas específicas.

- Analisar e avaliar o código: realizar uma avaliação detalhada da dos códigos Java produzidos pelas IA, examinando como a compreensão do significado do código se compara aos padrões humanos e identificar diferenças e semelhanças em relação aos códigos escritos por humanos.

- Adaptar o teste de Turing: Adaptar o teste de Turing para a avaliação de códigos-fonte em Java, de modo a criar um método eficaz para determinar se o código foi gerado por uma IA ou por um programador humano.

- Realizar experimentos práticos: implementar experimentos práticos nos quais alunos do CI-UFPB avaliam códigos gerados por IA e códigos humanos, utilizando os critérios heurísticos abordados e o teste de Turing adaptado.

- Analisar os resultados: analisar os resultados dos experimentos, examinando se os avaliadores foram capazes de distinguir de forma precisa entre os códigos gerados por IA e códigos humanos, e identificar quais métricas e critérios heurísticos foram mais eficazes na identificação da origem do código.

- Formular recomendações práticas para a indústria de TI: a partir da análise dos resultados, desenvolver diretrizes específicas que possam ser aplicadas na prática de programação assistida por IA. Estas recomendações visam otimizar a eficiência e a qualidade na geração de código por IA. Além disso, identificar e discutir os impactos dessas práticas na indústria e na comunidade acadêmica.

1.4. Estrutura do Trabalho

Este trabalho é formado por cinco capítulos. O primeiro capítulo traz a introdução à área estudada, juntamente com o problema analisado durante a pesquisa e seus objetivos. O segundo capítulo é formado pelos Conceitos Gerais e Revisão da Literatura. O terceiro capítulo é composto pela Metodologia. O quarto capítulo é composto pela Apresentação e Análise dos Resultados. O quinto e último capítulo é composto pela conclusão e trabalhos futuros.

2. CONCEITOS GERAIS E REVISÃO DA LITERATURA

Este capítulo busca oferecer uma revisão aprofundada dos conceitos fundamentais e do estado da arte existente no campo de estudo, abordando as métricas, critérios heurísticos e teorias relevantes para identificar a origem de códigos gerados por IA. A literatura existente aborda diversos tópicos, desde critérios heurísticos específicos até adaptações do teste de Turing para avaliar a capacidade das IA em produzir código indistinguível da criação humana.

O notável progresso na IA nas últimas décadas remodelou profundamente vários aspectos da vida moderna, impactando vários campos, desde a saúde até o entretenimento [11,12]. Na vanguarda desta revolução estão os LLMs (*Large Language Models* - Modelos de Linguagem de Grande Escala), um subconjunto de IA que transformou fundamentalmente a geração de linguagem. A evolução dos LLMs remonta ao desenvolvimento de algoritmos de ML (*Machine Learning* - Aprendizado de Máquina) que poderiam ser treinados para prever pontos de dados futuros com base nos padrões identificados nos dados originais.

No entanto, a mudança transformadora no campo veio com o DL (*Deep Learning* - Aprendizado Profundo), uma classe de modelos de ML projetados para aprender de forma automática e adaptativa estruturas complexas a partir de dados de alta dimensão. Esses avanços culminaram na criação de LLMs sofisticados como o GPT [13], capazes de gerar texto semelhante ao humano que é quase indistinguível do trabalho de um autor humano. Os LLMs aproveitam DL, feedback humano e vastos conjuntos de dados para compreender e gerar texto semelhante ao humano. Esses modelos, treinados em diversas entradas de texto, podem prever palavras subsequentes em uma sequência, gerando assim sentenças coerentes e contextualmente relevantes. O GPT-3, com seus 175 bilhões de parâmetros, pode realizar uma variedade de tarefas linguísticas, como tradução, resposta a perguntas e geração de texto, sem treinamento específico orientado a tarefas [14].

No entanto, o que diferencia os LLMs é a sua capacidade de gerar conteúdo criativo, como histórias ou ensaios, o que é uma tarefa complexa que requer uma compreensão diferenciada da linguagem, além de poder gerar códigos-fonte. Sua capacidade de compreender o estilo, o tom e as sutilezas contextuais da linguagem os torna ferramentas poderosas para inúmeras aplicações, desde a criação de conteúdo até o atendimento automatizado ao cliente [15]. Apesar de suas habilidades impressionantes, os LLMs apresentam limitações. Ocasionalmente, eles podem gerar conteúdo sem sentido ou impróprio

e são altamente sensíveis às frases de entrada [16–18]. Estas limitações abrem áreas de investigação para melhorar ainda mais as capacidades e a segurança dos LLMs [19], impulsionando a procura de modelos de IA mais sofisticados.

Apesar desses avanços significativos, a diferenciação entre texto gerado por humanos e texto gerado por IA apresenta um conjunto único de desafios. A sofisticação dos LLMs atuais significa que eles podem imitar nuances estilísticas humanas, tornando seus resultados mais difíceis de distinguir [20]. Considerações éticas também surgem, à medida que o uso indevido desses modelos de IA para gerar conteúdo enganoso, ou prejudicial, se torna uma preocupação crescente [21]. Além disso, as implicações destes desafios no mundo real são significativas, com potenciais ramificações em campos como o jornalismo, a criação de conteúdos online e a integridade acadêmica.

Em meio às crescentes preocupações com a integridade acadêmica, a sofisticação e a acessibilidade dos geradores de texto baseados em LLM exigem a capacidade de distinguir entre texto gerado por humanos e texto gerado por IA, já que o potencial uso indevido dessas tecnologias ameaça minar os princípios fundamentais de originalidade e esforço individual do sistema educacional. O advento dos LLMs apresenta desafios substanciais para manter a integridade acadêmica. Primeiro, sua capacidade de gerar textos contextualmente relevantes e de alta qualidade possibilita que os alunos usem essa tecnologia para concluir tarefas ou trabalhos, em vez de escrever conteúdo original [22]. Isto poderia levar a um aumento preocupante do plágio e da desonestidade intelectual, ao submeter trabalhos que não são produto da compreensão ou do esforço do indivíduo. Além disso, a base de conhecimento ampla e diferenciada que estes LLMs possuem pode potencialmente superar as capacidades humanas em várias tarefas acadêmicas. Por exemplo, um aluno poderia aproveitar um gerador de texto de IA para escrever um código fonte sobre um tópico que mal entende, ignorando totalmente o processo de aprendizagem. Isto perturba fundamentalmente o objetivo da educação, que é desenvolver o conhecimento, a compreensão e as capacidades de pensamento crítico de uma pessoa.

Finalmente, a sofisticação destes modelos também poderia comprometer os métodos de detecção que as instituições de ensino empregam para identificar má conduta acadêmica. As ferramentas tradicionais de detecção de plágio podem não identificar o texto gerado por IA como plagiado [23], uma vez que gera conteúdo exclusivo baseado em padrões escritos por humanos. Assim, embora os LLMs ofereçam possibilidades interessantes para muitas aplicações, eles também ameaçam a integridade acadêmica [24,25]. Os educadores, as instituições e os criadores de IA devem compreender e enfrentar estes desafios,

desenvolvendo novas estratégias e ferramentas para preservar os princípios de honestidade e originalidade no mundo acadêmico

2.1. Linguagem de programação Java

Uma linguagem de programação é um conjunto de instruções e regras que permite aos programadores comunicar-se com computadores para criar programas de software [26]. Linguagens de programação são usadas para desenvolver aplicativos e sistemas, oferecendo uma estrutura para dar comandos e construir a lógica necessária para que computadores executem tarefas específicas [26], as linguagens de programação consiste em palavras-chave, sintaxe e estruturas lógicas que facilitam a criação de algoritmos e o controle do fluxo de execução de um programa [27].

Essas podem ser classificadas de várias formas, como de baixo ou alto nível [26], dependendo do quanto se aproximam da linguagem humana em comparação ao código de máquina. As linguagens de baixo nível, como Assembly, oferecem comandos que operam quase diretamente no hardware, enquanto as de alto nível, como Python, Java e C++, são mais próximas da linguagem humana e são independentes da arquitetura da máquina [27].

Linguagens de o também podem ser classificadas como interpretadas e compiladas [26]. Linguagens compiladas, como C e C++, são convertidas em código de máquina antes da execução, enquanto linguagens interpretadas, como Python e JavaScript, são traduzidas em tempo real [26]. Isso influencia a eficiência e portabilidade dos programas, uma consideração importante para desenvolvedores que escolhem a linguagem adequada para seu projeto [27].

Java é uma linguagem de programação orientada a objetos, criada pela Sun Microsystems em 1995 e adquirida pela Oracle em 2010 [30]. Ela é conhecida por ser robusta, segura e independente de plataforma, características que a tornaram uma das linguagens mais populares no desenvolvimento de aplicativos empresariais, aplicativos Android e sistemas distribuídos [28]. A promessa de "escrever uma vez, rodar em qualquer lugar" é uma das maiores vantagens do Java, pois permite que o mesmo código seja executado em diferentes sistemas operacionais por meio da Java Virtual Machine (JVM) para ser de fácil leitura e manutenção, promovendo práticas de codificação padronizadas e bem documentadas [28]. Em Java, desenvolvedores humanos tendem a seguir convenções de nomenclatura, como a capitalização de nomes de classes e o uso de camelCase para métodos e variáveis, além de

priorizar a modularidade e o encapsulamento para manter o código claro e organizado [29]. Além disso, o uso de comentários é incentivado para explicar blocos complexos, especialmente quando a lógica não é intuitiva, o que auxilia na colaboração entre equipes e facilita a manutenção do software [28].

Essas práticas ritos manualmente, ajudam a criar um padrão de legibilidade e organização que muitas vezes diferencia o código humano do gerado por IA, pois o desenvolvimento humano tende a focar na clareza e na estruturação pensada para outros programadores [28]. Esses aspectos facilitam o entendimento do código e tornam a manutenção mais prática, além de reforçar o controle de qualidade ao longo do ciclo de vida do software.

Em Java, espera-se que um código desenvolvido por um programador humano tenha características como legibilidade, clareza, modularidade e uso consistente de convenções [29]. Em termos de legibilidade, variáveis e métodos devem ter nomes descritivos e intuitivos, facilitando o entendimento do propósito de cada parte do código sem a necessidade de extensos comentários explicativos [29]. A clareza do código Java também é reforçada pela organização lógica, na qual métodos e classes são estruturados de forma modular [30], ou seja, com cada componente cumprindo uma função específica e clara dentro do programa fator importante é o uso de convenções de nomenclatura e de formatação, como a capitalização de nomes de classes e o uso de camelCase para variáveis e métodos [29]. Isso facilita a colaboração entre programadores e a manutenção do código a longo prazo. Além disso, um código humano geralmente demonstra uma estrutura otimizada e cuidadosa no tratamento de exceções, bem como o uso apropriado de estruturas de dados para atender ao desempenho e à funcionalidade desejada [29].

Com código gerado por IA pode exibir um uso mais genérico de nomes de variáveis e métodos, além de uma menor flexibilidade em ajustar a estrutura do programa para requisitos específicos [29]. A presença de práticas de programação comuns, como o uso de interfaces e herança adequada [29], também é outro sinal que sugere que o código foi elaborado por um humano experiente em Java.

2.2. Inteligências Artificiais Usadas

2.2.1. ChatGPT

O ChatGPT, assim como outros modelos da OpenAI, é baseado em uma arquitetura de deep learning conhecida como Transformer [31]. A arquitetura Transformer, revolucionou o processamento de linguagem natural (*Natural Language Processing* - NLP) ao permitir que redes neurais processem dados em paralelo, utilizando mecanismos de "atenção" para priorizar informações contextuais [31]. Esses mecanismos de atenção, em particular a "self-attention," ajudam o modelo a entender a relevância de cada palavra em uma sequência, o que é fundamental para gerar respostas coesas e contextualizadas.

A arquitetura do ChatGPT se baseia em camadas empilhadas de "attention heads" que ajudam a capturar relações complexas entre palavras e conceitos [31]. Cada camada do Transformer possui dois componentes principais: o bloco de atenção e o feed-forward neural network [31]. Esses blocos de atenção examinam todas as partes do input de uma vez, ajustando-se com base nas relações de dependência que o modelo aprende entre as palavras [31]. Esse processo se repetido em múltiplas camadas aumenta a capacidade do modelo de compreender contextos mais complexos [31].

A OpenAI treinou o ChatGPT em uma grande quantidade de texto extraído da internet, incluindo websites, livros e artigos [32]. Esse treinamento é conhecido como "aprendizagem não supervisionada", onde o modelo aprende as estruturas linguísticas e padrões sem que as respostas corretas sejam fornecidas de antemão. Após essa etapa, é realizada a chamada "fine-tuning supervisionada", onde exemplos específicos e feedback humano ajudam o modelo a refinar suas respostas, tornando-as mais úteis e menos propensas a erros.

Durante o treinamento, o ChatGPT passa por uma etapa chamada de "ajuste de alinhamento", onde os desenvolvedores utilizam técnicas como a "aprendizagem por reforço com feedback humano" (RLHF) [31]. Essa técnica envolve a coleta de feedback humano sobre as respostas do modelo para ajustar seu comportamento, visando produzir respostas mais seguras e relevantes para o usuário. O RLHF permite que o modelo aprenda a evitar respostas potencialmente prejudiciais ou irrelevantes [31].

Por fim, a implementação técnica do ChatGPT se baseia em infraestruturas de alto desempenho, com o modelo executado em GPUs (unidades de processamento gráfico) otimizadas para cargas de trabalho de deep learning [31]. Essas GPUs processam grandes volumes de dados, garantindo a velocidade e a precisão nas respostas que você vê no chat. A

quantidade de camadas e parâmetros que o modelo possui influencia diretamente sua capacidade de lidar com tarefas complexas de linguagem.

2.2.2. Bard

O Bard ou Gemini do Google é um modelo de IA multimodal altamente avançado desenvolvido pela DeepMind, projetado para lidar com uma ampla gama de tipos de dados e tarefas simultaneamente [34]. Ao contrário dos modelos tradicionais que se concentram apenas em texto, o Bard integra recursos para entender e gerar conteúdo de texto, imagens, áudio e vídeo [33]. Isso o torna uma ferramenta mais versátil em comparação com modelos como o GPT-4, que se concentra principalmente no processamento de texto. A arquitetura por trás do Bard aproveita um modelo transformador, semelhante aos usados em outros grandes modelos de IA, mas sua verdadeira inovação está em seu treinamento simultâneo em vários formatos de mídia, permitindo que ele entenda o contexto de uma forma mais sutil [34].

Um dos recursos de destaque do Bard é sua capacidade de processar e integrar dados de vários tipos de entradas, como combinar texto e imagens ou interpretar gráficos e suas legendas em uníssono [34]. Essa abordagem multimodal aprimora seus recursos de raciocínio, permitindo que ele estabeleça conexões entre diferentes tipos de conteúdo. O Bard também ostenta uma impressionante janela de contexto longa, particularmente em sua versão 1.5 Pro, que permite processar até dois milhões de tokens em um único prompt — ideal para lidar com documentos grandes e complexos.

A família Bard é estruturada em torno de uma rede de modelos que colaboram, tornando-a altamente flexível para diversas aplicações em vários campos, incluindo escrita criativa, resolução de problemas e tomada de decisões [33]. Ela está disponível em várias versões, incluindo uma versão gratuita que é um modelo premium chamado Bard Ultra, oferecendo recursos mais avançados para uso profissional [33]. O Bard é integrado a vários serviços do Google, como Gmail e Docs, e também pode ser acessado por meio de aplicativos móveis, aumentando sua utilidade em diferentes plataformas [34].

2.3. Critérios Heurísticos na Identificação de Códigos Fonte

Os critérios heurísticos desempenham um papel crucial na diferenciação entre códigos gerados por IA e códigos humanos[35]. Elementos como formatação [2], número de linhas [35], comentários [2], átomos de confusão [4] e nome de identificadores [5] têm sido extensivamente explorados.

A formatação de código-fonte refere-se à maneira como o código é organizado, estruturado e apresentado, garantindo que seja legível, consistente e fácil de entender por humanos [2]. Embora a formatação não altere a execução ou a lógica do código, ela facilita o trabalho dos programadores [2].

O número de linhas em um código-fonte refere-se à quantidade de linhas escritas por um programador para implementar a lógica de um programa [35]. Essa contagem inclui linhas que contêm código executável, bem como espaços em branco e estruturas organizacionais, como quebras de linha para legibilidade [35]. O número de linhas de código LOC (*Lines of Code*) é frequentemente utilizado como uma métrica para avaliar o tamanho de um programa, a complexidade de um projeto, e a produtividade do desenvolvimento [35].

Comentários em código-fonte são anotações feitas pelos programadores dentro do código para explicar, ou descrever partes específicas da lógica, sem afetar a execução do programa [2]. Eles são ignorados pelo compilador, ou interpretador, durante a execução do código [39], sendo destinados exclusivamente à leitura por humanos. Os comentários podem ser de linha única, ou múltiplas linhas, dependendo da linguagem e da extensão da explicação [2].

Ao escrever código, os desenvolvedores comunicam sua intenção a outros desenvolvedores. Interpretar corretamente sua intenção é crucial para os processos de manutenção e evolução do software [4]. Essa interpretação ocorre por meio de um processo de compreensão de código, no qual um desenvolvedor lê o código-fonte, geralmente escrito por outro desenvolvedor, e entende seu comportamento. No entanto, a interpretação do desenvolvedor de um pedaço de código pode frequentemente se diferenciar daquela de quem escreveu o código, devido a pequenos padrões que podem causar mal-entendidos. Esses pequenos padrões, que podem ofuscar o código e confundir os desenvolvedores, fazendo com que eles julguem mal o comportamento do código são chamados de átomos de confusão [37,38,46].

Nome de identificadores ou Identificadores em um código-fonte são nomes dados a elementos de um programa, como variáveis, funções, classes, métodos e constantes [5]. Esses identificadores são utilizados para referenciar e acessar esses elementos ao longo do código. Eles são essenciais para a compreensão e manipulação do código por humanos e máquinas [36]. Cada linguagem de programação tem regras específicas para a nomeação de identificadores, como restrições de caracteres permitidos e a não utilização de palavras-chave

reservadas. Estas métricas são empregadas em diversas situações para compreender a origem dos códigos gerados, buscando estabelecer critérios que diferenciem as produções das IA das criações humanas.

2.4. O Teste de Turing Adaptado na Avaliação de Código Fonte

O Teste de Turing foi proposto em 1950 pelo matemático e cientista da computação britânico Alan Turing. O teste é um experimento mental que busca responder à pergunta "As máquinas podem pensar?" e é amplamente considerado um marco na história da inteligência artificial [1]. O objetivo do teste é verificar se uma máquina pode exibir um comportamento inteligente indistinguível de um ser humano [1].

O Teste de Turing é uma avaliação em que uma máquina tenta imitar o comportamento humano durante uma conversa. Ele envolve três participantes:

1. Um humano (o "interrogador" ou avaliador).
2. Outro humano.
3. Uma máquina (geralmente, um programa de computador ou sistema de IA).

Todos se comunicam por meio de uma interface, que não revela a identidade física dos participantes, geralmente por texto, para que o interrogador não saiba quem é a máquina e quem é o humano. O interrogador faz perguntas a ambos os participantes, e a tarefa é determinar qual dos dois é a máquina e qual é o humano, apenas com base nas respostas.

Se a máquina conseguir responder de tal forma que o interrogador não possa distingui-la do humano, ela passa no Teste de Turing, sendo considerada "inteligente", de acordo com o critério proposto por Turing. Abaixo pode-se ver um exemplo na Figura 1 de como funciona o fluxo do teste de turing

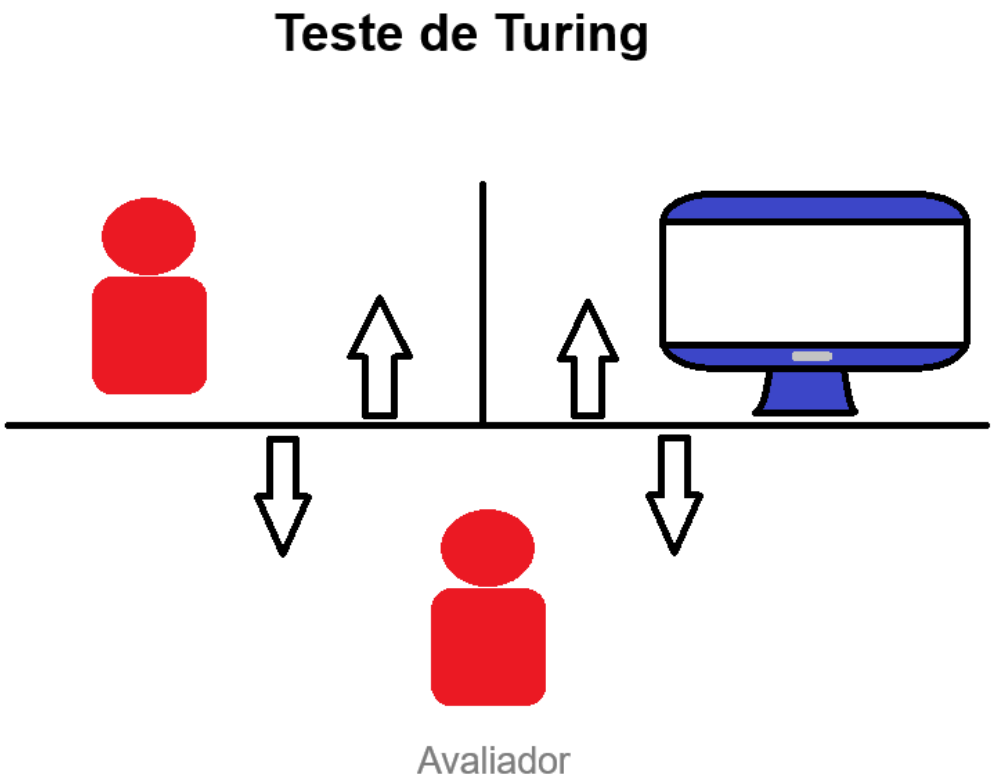


Figura 1: Exemplo de fluxo do Teste de Turing

Estrutura do Teste:

- Interrogador (humano): Faz perguntas para a máquina e para o outro humano, com o objetivo de identificar qual é qual.
- Máquina (inteligência artificial): Tenta responder, de forma que o interrogador não consiga identificá-la como máquina.
- Humano: Responde às perguntas com o objetivo de se comportar normalmente, mas sem revelar sua identidade claramente ao interrogador.

O teste é considerado bem-sucedido se, ao final da interação, o interrogador não conseguir discernir corretamente a identidade da máquina e do humano, com uma precisão maior do que o acaso (50%).

O Teste de Turing não exige que a máquina seja consciente, ou entenda as perguntas que recebe. Ele foca unicamente na capacidade da máquina de simular respostas humanas convincentes, dentro de um determinado período de tempo. Ou seja, o teste não avalia

diretamente a compreensão da máquina sobre o que está sendo dito, mas sim a sua habilidade de se passar por um humano em uma conversa natural.

Esse teste foi adaptado para avaliar a capacidade das IA em produzir códigos-fonte que se assemelham a criações humanas. Esta adaptação do teste de Turing torna-se uma ferramenta fundamental para avaliar a capacidade das IA em produzir códigos que possam confundir os avaliadores humanos [1].

3. METODOLOGIA

Considerando a natureza do estudo que investiga a distinção entre códigos gerados por IA e códigos escritos por humanos, uma abordagem quantitativa se mostra apropriada para alcançar os objetivos propostos, para isso foi desenvolvido um questionário, cujas perguntas se encontram no Google Forms [40] para facilitar a distribuição para os alunos do CI.

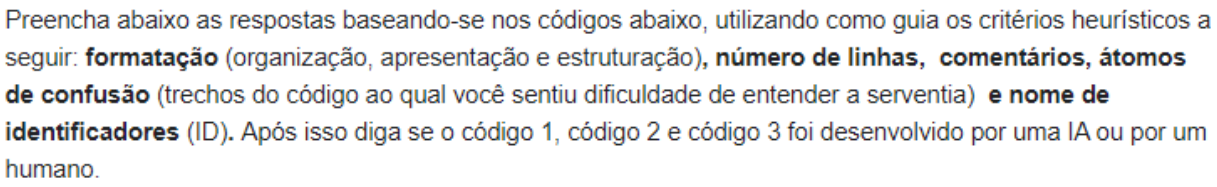

Preencha abaixo as respostas baseando-se nos códigos abaixo, utilizando como guia os critérios heurísticos a seguir: **formatação** (organização, apresentação e estruturação)**, número de linhas, comentários, átomos de confusão** (trechos do código ao qual você sentiu dificuldade de entender a serventia) **e nome de identificadores** (ID)**.** Após isso diga se o código 1, código 2 e código 3 foi desenvolvido por uma IA ou por um humano.

Figura 2: Critérios Heurísticos

A coleta de dados consiste na obtenção de conjuntos de códigos-fonte em Java. Os conjuntos de dados serão compostos por 3 códigos, 2 gerados por IA [7][8] e 1 código escrito por humano [6], que podem ser encontrados no Apêndice do texto e foram gerados através de um comando ou *prompt* simples que diz: “gere um analisador léxico na linguagem Java” tanto no ChatGPT quanto no Bard. Como dito os códigos tratam-se de ser um compilador para fazer a análise léxica, que nada mais é o processo de converter uma sequência de caracteres em uma sequência de *tokens*. Esses códigos foram desenvolvidos para refletir as características estruturais de códigos escritos por humanos e por IA. O código 1 [6], código 2 [7] e código 3 [8].

Foram estabelecidas métricas específicas e critérios heurísticos para avaliação dos conjuntos de códigos. Isso inclui os elementos formatação [2], número de linhas [35], comentários [2], átomos de confusão [4] e nome de identificadores [5]. As métricas numéricas e critérios definidos serão aplicados a cada conjunto de códigos, para quantificar suas características.

O teste de Turing foi adaptado para a avaliação dos códigos. Foram preparados 3 conjuntos de códigos [6][7][8], de tal maneira que os avaliadores (alunos do CI-UFPB, com mais de 3 anos de experiência na área como pode ser visto na tabela 1) não tinham conhecimento prévio sobre a origem de cada código e foram orientados a realizar suas

análises baseando-se unicamente nas características do código, na Figura 3 pode-se ver um exemplo das perguntas feitas para os avaliadores.

Participantes	**Gênero**	**Idade**	**Período**	**Quantidade disciplinas de programação**	**Experiência com programação (em anos)**	**Quais linguagens domina**	**Possui experiência em java?**
Participante 1	Masculino	23 anos	P7	22 disciplinas	6 anos	C, Python, Java, Javascript	Sim
Participante 2	Masculino	22 anos	P8	18 disciplinas	5 anos	Python	Sim
Participante 3	Masculino	20 anos	P5	16 disciplina	3 anos	Java, Python	Sim
Participante 4	Masculino	20 anos	P4	14 disciplinas	3 anos	Java, Javascript	Sim
Participante 5	Feminino	23 anos	P7	18 disciplinas	5 anos	C, Python	Sim
Participante 6	Masculino	25 anos	P8	22 disciplinas	6 anos	Java, python, Go	Sim
Participante 7	Masculino	23 anos	P7	17 disciplinas	5 anos	C, C++	Sim
Participante 8	Masculino	21 anos	P7	19 disciplinas	5 anos	Python, Java	Sim
Participante 9	Masculino	23 anos	P7	20 disciplinas	5 anos	C++, Html	Sim
Participante 10	Masculino	22 anos	P7	20 disciplinas	5 anos	Html, Java	Sim

Tabela 1: Perfil demográfico participantes

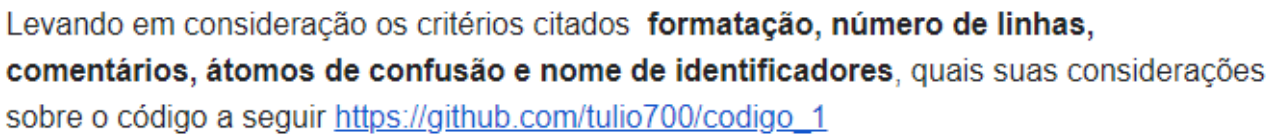


Figura 3: Exemplo de pergunta feita para análise do código

De maneira a analisar e entender quais aspectos foram utilizados para chegar a conclusão de qual a origem do código inicialmente, foram feitas três perguntas levando em consideração os critérios heurísticos usados, uma para cada um dos códigos. Após isso foi feita uma pergunta para entender se eles chegaram a utilizar mais algum outro critério que não os citados anteriormente, se foi utilizado alguma outra coisa como: ferramenta externa, sites ou até mesmo outra IA, qual a origem do código 1, código 2 e código 3, além de perguntar se

os critérios ajudaram ou não na conclusão se os códigos foram gerados por IA e dizer o motivo deles terem ajudado ou não nessa decisão.

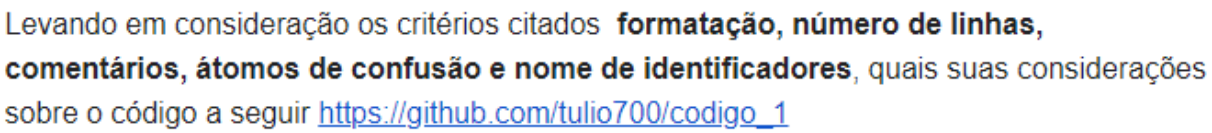


Figura 4: Pergunta feita para análise do código 1

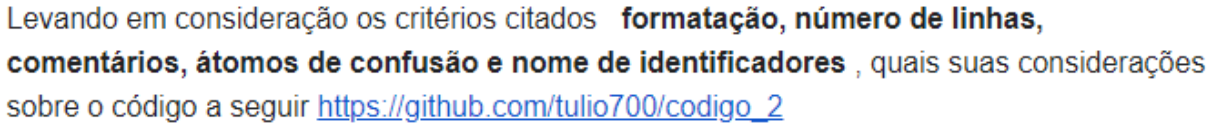


Figura 5: Pergunta feita para análise do código 2

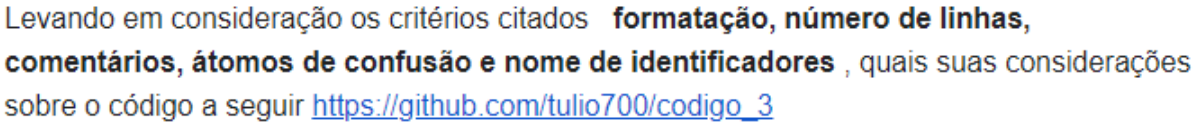


Figura 6: Pergunta feita para análise do código 3

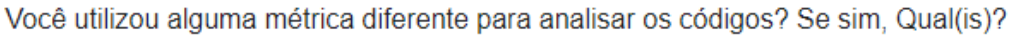


Figura 7: Pergunta feita para análise de alguma métrica diferente da apresentada

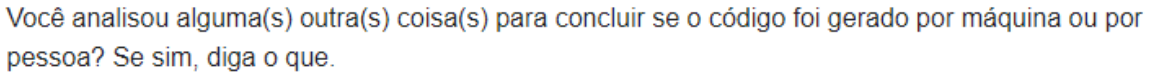


Figura 8: Pergunta feita para análise de algum outro método utilizado

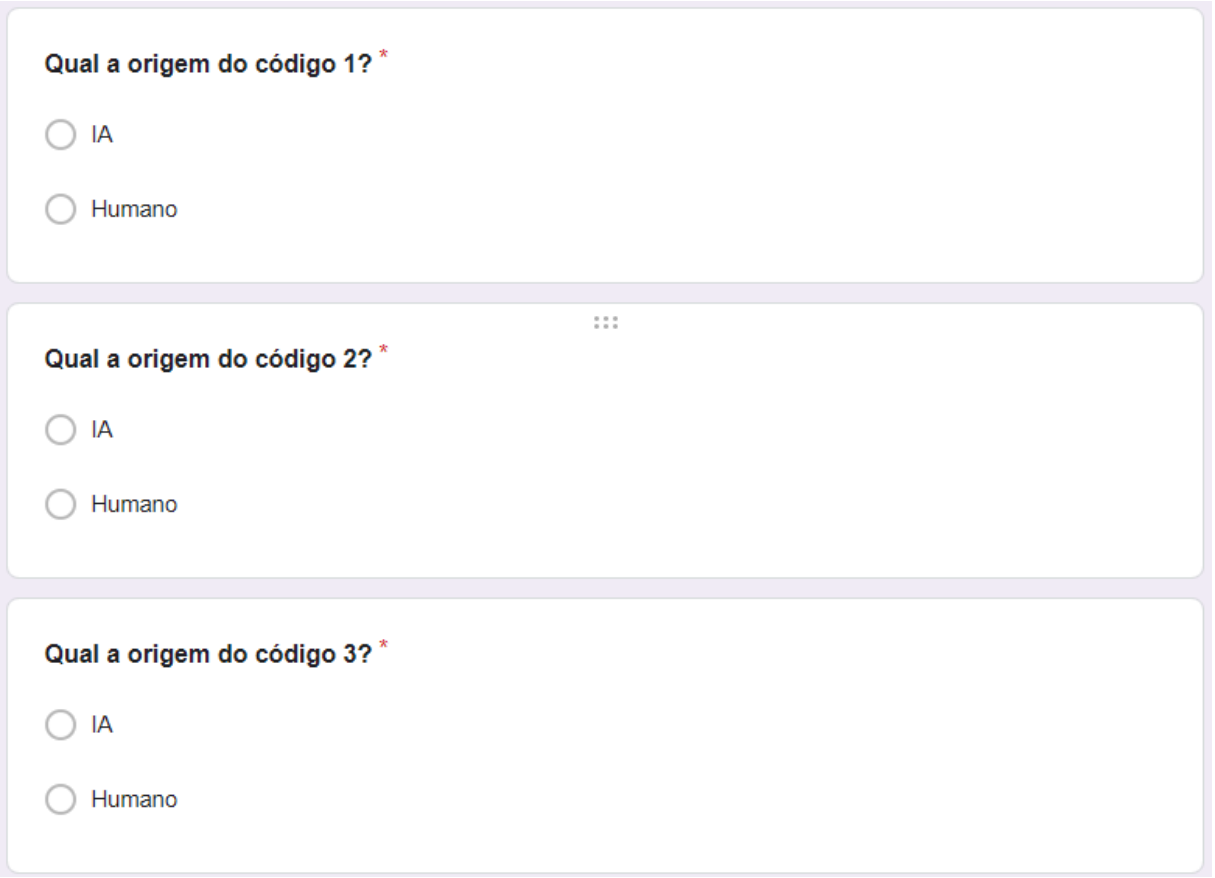


Figura 9: Perguntas feitas para análise da origem de cada código

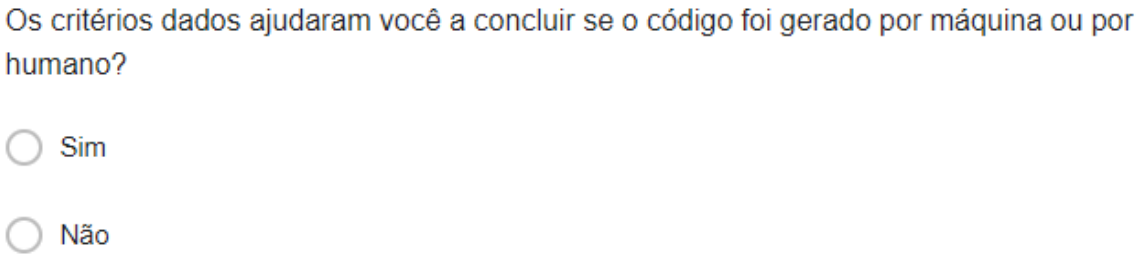


Figura 10: Pergunta feita para entender se as métricas ajudaram a descobrir a origem do código

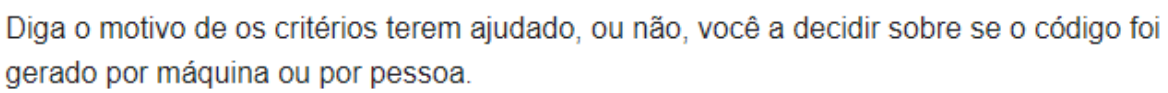


Figura 11: Pergunta feita para entender como as métrica ajudaram no processo

Os resultados obtidos do teste de Turing adaptado foram submetidos a análises estatísticas. Após isso, foi analisada a porcentagem de acerto referente a cada código , e, para cada resposta certa, foram analisadas quais métricas o avaliador usou, assim comparando-as, para saber se existiu alguma correlação entre cada acerto.

4. APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Esta capítulo aborda os resultados obtidos da pesquisa, mostrando como as IA, como ChatGPT e Bard, foram capazes de simular códigos-fonte [7][8] para confundir os avaliadores, levando-os a acreditar que foram gerados por seres humanos.

4.1. Análise das perguntas levando em consideração os critérios heurísticos

Para entendermos o que cada pessoa fez para chegar a conclusão de se um código foi gerado ou não por IA, é interessante analisar separadamente cada pergunta. As respostas para cada uma das perguntas se encontram no Apêndice, além dos 3 códigos citados no texto.

4.1.1. Análise das respostas referentes à Pergunta 1

Analisando as respostas fornecidas, pode-se observar um padrão de críticas relacionadas à legibilidade e clareza do código [6]. O formato, embora variável, dependendo das percepções individuais, foi frequentemente relatado como problemático, com destaque para a falta de comentários, dificultando a compreensão para quem não está familiarizado com o código [6]. A questão dos identificadores, desde os longos aos curtos, também foi uma preocupação recorrente. Em termos de número de linhas, muitos consideram o código longo, o que segundo Martin [39], pode ser prejudicial à legibilidade e manutenção, embora o comprimento do código nem sempre seja visto diretamente como um problema, dependendo do contexto.

No entanto, houve divergências entre as respostas. Alguns acharam o código bem formatado, enquanto outros notaram blocos aninhados que dificultavam a leitura. Além disso, embora algumas respostas indicassem que o número de linhas não era necessariamente um problema, outras destacaram que o código poderia ter sido mais simples. As opiniões sobre os identificadores também variam: alguns acreditam que estão conforme as melhores práticas, enquanto outros citaram palavreado excessivo ou abreviaturas.

Concluindo, as principais críticas podem ser resumidas na necessidade de maior clareza e otimização do código, o que vai ao encontro de McConnell [41], que defende que o código deve ser escrito para facilitar a legibilidade e a manutenção, e não apenas para ser

executado... A falta de comentários e muitos blocos aninhados dificultam a compreensão, principalmente para leitores que não estão familiarizados com a linguagem, ou com o projeto específico. Melhorar esses aspectos poderia tornar o código mais acessível e eficiente.

4.1.2. Análise das respostas referentes à Pergunta 2

Na análise das respostas fornecidas no Código 2 [7], observa-se um padrão geral de melhoria em relação ao código anterior [6]. O formato tem sido frequentemente elogiado por ser mais organizado e legível, com nomes de identificação mais apropriados e funções mais claras. O número de linhas foi reduzido, o que, segundo Martin [39], auxilia na manutenção e clareza do código. Porém, a ausência ou escassez de comentários representou mais uma vez um ponto crítico, dificultando o entendimento para quem não conhecia previamente o código, conforme as boas práticas sugeridas por McConnell [41]. As principais divergências entre os testadores dizem respeito à percepção da complexidade do código. Enquanto alguns disseram que o código era mais compacto e simplificado, outros disseram que o aumento da complexidade introduziu mais confusão, o que poderia dificultar a compreensão, especialmente para quem não está familiarizado com Java. Além disso, houve divergências quanto à origem do código, com alguns sugerindo que ele foi gerado por IA, devido à falta de preenchimento em uma das estruturas de controle.

Concluindo, o Código 2 [7] apresenta uma melhoria em termos de clareza e legibilidade em relação ao Código 1 [6], com nomes de identificação mais apropriados e melhor formatação. Porém, segundo McConnell [41], a falta de comentários continua sendo uma falha grave, o que reforça a necessidade de documentar adequadamente o código para facilitar o entendimento e a manutenção. O aumento da complexidade também precisa ser gerenciado com cuidado para evitar átomos confusos, que podem prejudicar a legibilidade, algo que Martin [39] critica quando se trata de código sustentável e compreensível. No entanto, houve divergências entre as respostas. Alguns acharam o código bem formatado, enquanto outros notaram blocos aninhados que dificultavam a leitura. Além disso, embora algumas respostas indicaram que o número de linhas não era necessariamente um problema, outras destacaram que o código poderia ter sido mais simples. As opiniões sobre os identificadores também variam: alguns acreditam que cumprem as melhores práticas, enquanto outros citaram palavreado excessivo ou abreviaturas.

Concluindo, as principais críticas podem ser resumidas na necessidade de maior clareza e otimização do código, o que vai ao encontro de McConnell [41], que defende que o código deve ser escrito para facilitar a legibilidade e a manutenção, e não apenas para ser executado... A falta de comentários e muitos blocos aninhados dificultam a compreensão, principalmente para leitores que não estão familiarizados com a linguagem, ou com o projeto específico. Melhorar esses aspectos poderia tornar o código mais acessível e eficiente.

4.1.3. Análise das respostas referentes à Pergunta 3

Na análise das respostas para o Código 3 [8], os avaliadores identificaram que o código apresenta uma melhora em relação aos anteriores, especialmente na clareza e legibilidade, devido ao uso de um *switch* em vez das funções "ifs" repetidas, o que está em linha com as boas práticas sugeridas por Martin [39] para melhorar a legibilidade e a manutenção do código. A formatação também foi amplamente elogiada, com a indentação sendo considerada adequada e os nomes dos identificadores claros. No entanto, a ausência de comentários continua a ser uma crítica constante, o que, segundo McConnell [41], compromete o entendimento, especialmente para programadores que não estão familiarizados com o código.

Houve algumas divergências entre os avaliadores. Enquanto alguns consideraram o código mais compacto e bem estruturado, outros ainda observaram a presença de átomos de confusão, especialmente devido à complexidade da lógica apresentada, que exigiria tempo para ser completamente compreendida. Além disso, surgiram dúvidas quanto à autoria do código, com alguns avaliadores acreditando que poderia ter sido gerado por uma IA, uma crítica que também foi levantada em relação aos códigos anteriores.

Em conclusão, o Código 3 [8] representa um equilíbrio entre os Códigos 1 e 2 [6][7], com melhorias na legibilidade e estruturação, como a substituição das funções "ifs" por uma função "switch" e a quebra de expressões booleanas longas. No entanto, a ausência de comentários continua a ser uma limitação significativa, reforçando a importância de seguir as boas práticas de documentação para facilitar a manutenção, como destacado por McConnell [41]. O uso de nomes claros para identificadores e a compactação do código também ajudam a torná-lo mais compreensível, embora ainda haja espaço para melhorar a clareza geral, principalmente com a inclusão de comentários.

4.2. Análise das perguntas buscando entender quais outros critérios ou ferramentas foram utilizadas

Ao correlacionar as respostas anteriores com as informações fornecidas com as respostas da Figura 7 e Figura 8, observa-se que a maioria dos participantes utilizou apenas sua experiência pessoal ou conhecimento de programação como critério principal para a avaliação dos códigos, sem o uso de métricas ou ferramentas específicas. Isso é consistente com os padrões de avaliação subjetiva encontrados nas respostas anteriores, em que as percepções individuais sobre legibilidade, formatação e clareza dos identificadores foram fortemente influenciadas por experiência e familiaridade com a linguagem Java.

No entanto, alguns participantes indicaram outros fatores ao avaliar a possibilidade de o código ter sido gerado por uma IA. O Participante 1, por exemplo, mencionou a maneira como os identificadores foram nomeados e a indentação como possíveis pistas, enquanto o Participante 4 citou a qualidade mediana que costuma associar aos códigos gerados por IA. O Participante 5 fez uma análise interessante, sugerindo que a redundância excessiva no primeiro código é um indício de que pode ter sido gerado por IA, especialmente se o *prompt* de entrada não foi bem estruturado, o que está em linha com estudos sobre IA generativa, que indicam que *prompts* mal formulados podem resultar em saídas subótimas [42]. Já o Participante 10 mencionou a presença de bibliotecas não nativas como um indicador, sugerindo que o comportamento de importação pode fornecer pistas sobre a origem do código.

A ausência de métricas quantitativas mais precisas, como a análise de complexidade ciclomática, ou ferramentas de detecção de padrões de IA, indica que os avaliadores confiaram em critérios subjetivos e qualitativos. De fato, estudos como o de Nagappan et al. [44] sugerem que métricas objetivas, como a complexidade do código, podem fornecer percepções mais claros sobre a qualidade e autoria de um código, especialmente em comparação com avaliações baseadas apenas na experiência.

As respostas refletem a importância da experiência e do conhecimento do avaliador para identificar a autoria do código. No entanto, a ausência de ferramentas formais e métricas quantitativas limita a precisão das análises. O uso de métricas objetivas, como a complexidade ciclomática [43], ou ferramentas específicas de detecção de padrões de IA, como os detectores de código gerados automaticamente, poderia complementar as percepções qualitativas e

fornecer uma base mais sólida para as conclusões. Além disso, a utilização de bibliotecas não nativas, ou a ausência de certos padrões de comentários, como mencionado pelo Participante 10, pode ser um indicativo útil, mas ainda carece de maior fundamentação em estudos empíricos, que abordem especificamente códigos gerados por IA, em comparação a humanos.

4.3. Análise da importância dos critérios abordados para descobrir a origem do código

Os critérios mencionados pelos entrevistados, como formatação, número de linhas, comentários, átomos de confusão e identificadores, mostraram-se úteis em diferentes níveis para determinar a origem dos códigos. Vários entrevistados destacaram a importância da clareza e padronização dos identificadores, algo frequentemente desconsiderado por IA's, como mencionado por aqueles que já utilizaram ferramentas de IA para geração de código (Resposta 1, Resposta 4, Resposta 9). Comentários e organização também foram vistos como indicadores-chave, com alguns participantes notando que códigos gerados por IA geralmente contêm comentários em excesso ou são bem comentados, enquanto humanos tendem a ter inconsistências nesse aspecto (Resposta 5, Resposta 6, Resposta 10).

Embora os critérios tenham sido considerados úteis, alguns entrevistados ressaltaram a necessidade de complementar esses critérios com a experiência pessoal. Por exemplo, participantes com mais contato com códigos gerados por IA conseguiram perceber padrões específicos, como redundância ou falta de otimização (Resposta 7), enquanto outros avaliaram aspectos como a qualidade da organização (Resposta 8). Em geral, os critérios ajudaram a identificar não apenas a possível autoria dos códigos, mas também pontos de melhoria, destacando como programadores podem se beneficiar dessas diretrizes para escrever código mais legível e eficiente (Resposta 3).

Por fim, a análise sugeriu que, embora os critérios abordados sejam úteis para identificar aspectos problemáticos e padrões de autoria, é a experiência do avaliador que refina essa percepção.

4.4. Análise de resultado referentes ao código 1

A análise do Código 1, considerando as respostas dos 10 entrevistados, revela uma compreensão significativa dos critérios que podem indicar a autoria de um código. Como pode ser visto pela Figura 12, 80% dos participantes identificou corretamente as limitações e características do código, destacando sua legibilidade deficiente, a falta de comentários e a complexidade excessiva, que são frequentemente associadas a códigos de iniciantes. As respostas apontaram que essas falhas podem ser indicativas de um código humano, particularmente de um programador novato, em contraste com a expectativa de que códigos gerados por IA possuam um padrão mais consistente e comentários que orientem a compreensão.

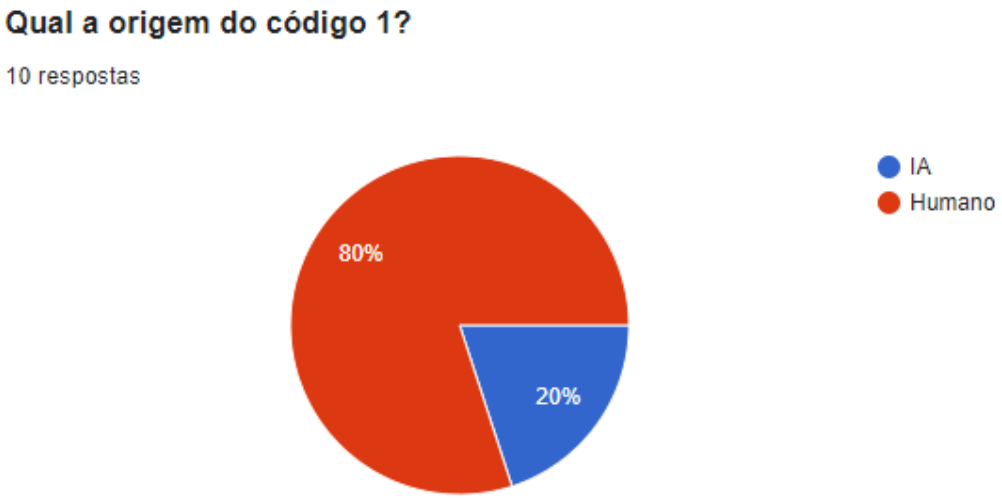


Figura 12: Respostas referentes a origem do código 1

Entretanto, os entrevistados 5 e 8 chegaram à conclusão incorreta de que o código 1 foi gerado por uma IA, evidenciando que a percepção sobre a autoria de um código pode ser influenciada por experiências pessoais e expectativas. Enquanto o Entrevistado 5 mencionou que a redundância era um erro típico de IA, a resposta de ambos sugere que os critérios que normalmente ajudam a discernir a origem do código podem ser mal interpretados em certos contextos. Isso ressalta a importância de uma análise mais profunda e crítica, que não apenas considere a aparência do código, mas também o contexto em que foi produzido.

Em conclusão, o código 1 exemplifica como a falta de experiência e conhecimento em práticas de codificação pode levar a uma qualidade inferior, que pode ser confundida com saídas de ferramentas de IA. A capacidade dos entrevistados de identificar os padrões

relevantes e a importância da experiência prévia são cruciais para fazer avaliações mais precisas sobre a autoria do código, sublinhando a complexidade do tema e a necessidade de um olhar atento e fundamentado na análise de códigos.

4.5. Análise de resultado referentes ao Código 2

A análise do Código 2, à luz das respostas dos 10 entrevistados, revela a complexidade envolvida na distinção entre códigos gerados por humanos e aqueles criados por ferramentas de IA. Embora a maioria dos participantes tenha percebido características que frequentemente indicam uma autoria humana, como a estrutura do código e a presença de alguns comentários, como pode ser visto na Figura 13, 40% dos entrevistados (2, 3, 5 e 8) chegaram à conclusão equivocada de que o código era humano. Essa confusão ressalta as sutilezas que podem existir nas saídas de IA, que, apesar de serem tecnicamente competentes, podem apresentar uma "qualidade mediana", que não se alinha perfeitamente com as expectativas tradicionais de programação humana.

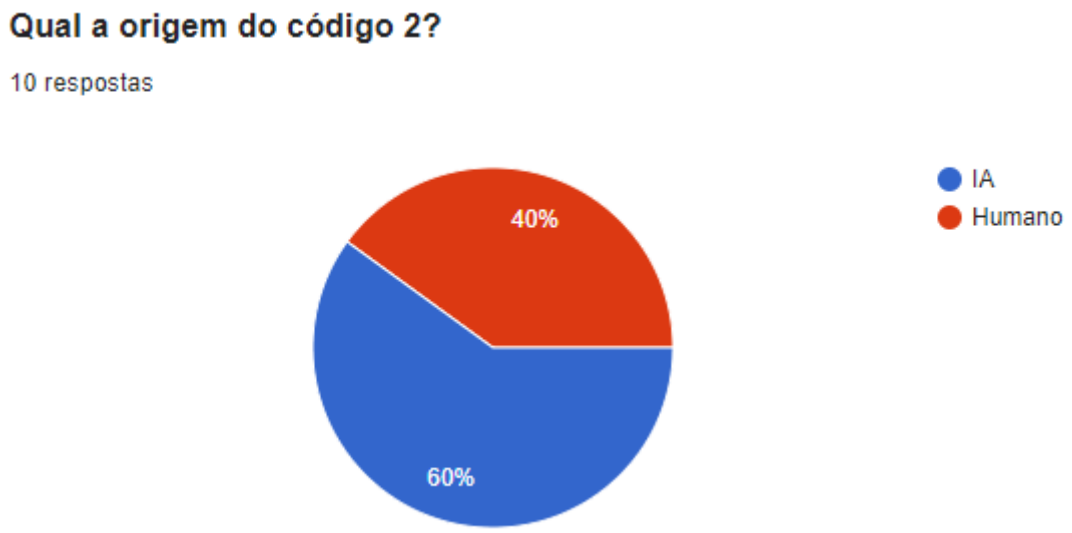


Figura 13: Respostas referentes a origem do código 2

Os entrevistados destacaram aspectos como a clareza dos identificadores e a legibilidade, que, embora sejam importantes, não são sempre suficientes para identificar a origem do código. Por exemplo, o Participante 5 mencionou que a "organização" e o "aspecto orgânico" do código sugerem autoria humana, sem considerar que IA's também podem produzir códigos com essa aparência, especialmente quando bem treinadas. A experiência de quem analisa o código e a familiaridade com padrões de IA são, portanto, fundamentais para uma avaliação mais acurada.

Essa situação sublinha a necessidade de uma análise mais profunda e baseada em critérios que vão além da aparência superficial. A capacidade de discernir a autoria de um código exige uma compreensão mais abrangente das práticas de codificação, incluindo as nuances da programação gerada por IA, que podem enganar até mesmo aqueles com alguma experiência no campo. Em resumo, o Código 2 exemplifica a dificuldade de identificação de autoria em um ambiente onde as ferramentas de IA estão se tornando cada vez mais sofisticadas, destacando a importância de uma análise crítica e uma familiaridade contínua com as tendências em programação e desenvolvimento de software.

4.6. Análise de resultado referentes ao código 3

A análise do Código 3 [8], considerando as respostas dos 10 entrevistados, revela as dificuldades em identificar a origem de códigos gerados por IA, mesmo entre aqueles com alguma experiência em programação. Apesar de 50% dos participantes (1, 3, 4, 5 e 10) erroneamente concluírem que o código foi produzido por um humano, como pode ser visto na Figura 14, o código em questão, na verdade, foi gerado por uma IA. Isso ilustra como a sofisticação das ferramentas de IA pode levar a confusões, especialmente quando o código gerado possui uma estrutura que pode parecer lógica e organizada.

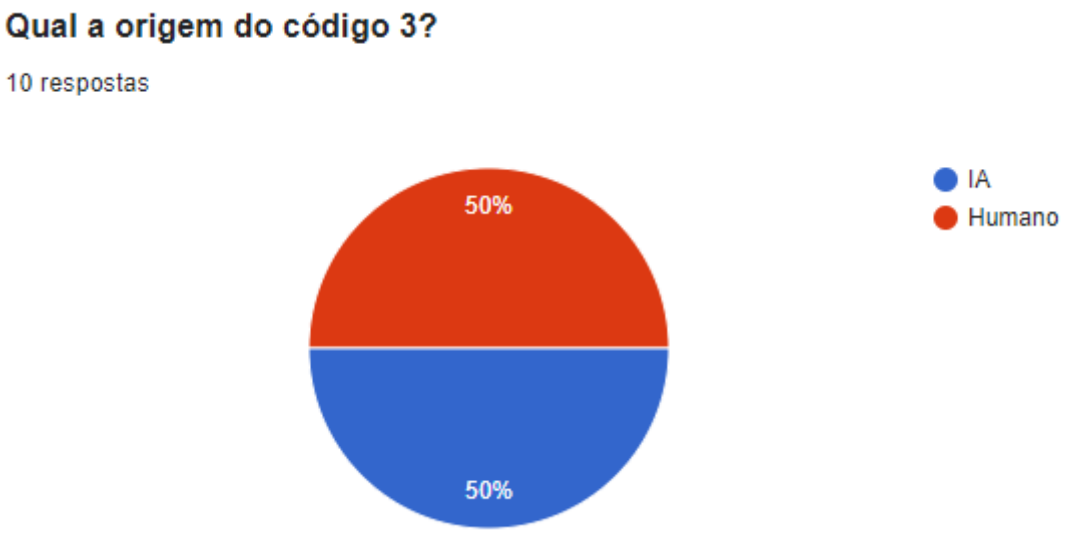


Figura 14: Respostas referentes a origem do código 3

Os entrevistados destacaram características como a legibilidade e a clareza dos identificadores, elementos que geralmente são esperados em códigos humanos. No entanto, essa percepção pode não ser suficiente para discernir a autoria, uma vez que ferramentas de IA estão cada vez mais capazes de produzir códigos que atendem a critérios de boa prática,

como formatação adequada e uma organização lógica. Por exemplo, o Participante 4 apontou melhorias no código, como a substituição de estruturas repetitivas por um switch, um sinal de sofisticação que poderia ser facilmente confundido com uma autoria humana.

A situação sublinha a importância de um olhar crítico e aprofundado na análise de códigos, enfatizando que características externas, como a estética e a legibilidade, podem não ser indicativas da origem real do código. É fundamental que os avaliadores desenvolvam uma compreensão mais ampla das capacidades das IA's na geração de código, pois isso pode levá-los a conclusões mais precisas e fundamentadas. Em resumo, o Código 3 exemplifica como a evolução das ferramentas de IA está desafiando as percepções tradicionais sobre a autoria de código, destacando a necessidade de uma análise contínua e Crítica em um campo em rápida evolução.

4.7. Importância dos critérios apresentados

Como pode ser visto na Figura 15, levando em consideração que 100% dos participantes afirmaram que os critérios fornecidos ajudaram na avaliação dos códigos, é possível afirmar que os critérios estabelecidos desempenham um papel fundamental na identificação da origem do código, seja gerado por IA ou por um humano. Esses critérios incluem aspectos como formatação, número de linhas, presença ou ausência de comentários, átomos de confusão e a clareza dos identificadores. No entanto, o resultado da pesquisa, com apenas 30% dos entrevistados acertando a origem dos códigos com exatidão, revela que, embora os critérios ajudem, eles não são suficientes por si só para garantir precisão.

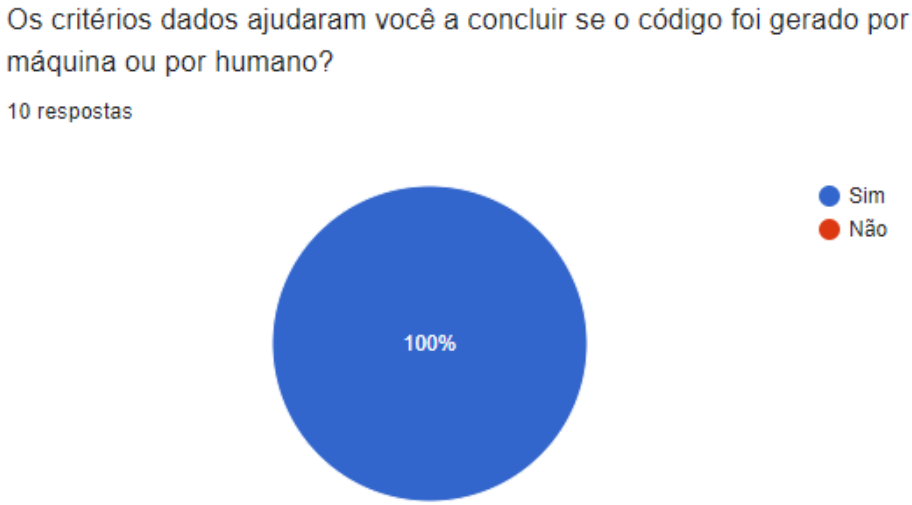


Figura 15: Gráfico importância dos critérios

A análise desses dados sugere que, embora esses critérios forneçam uma boa base, há nuances nos códigos gerados por IA, que podem passar despercebidas ou ser mal interpretadas. Isso ocorreu especialmente nos casos em que a IA produziu um código bem formatado e sem erros óbvios, confundindo os entrevistados e levando-os a acreditar que o código era humano. Por outro lado, os códigos gerados por humanos, que podem conter erros mais comuns ou inconsistências, também foram confundidos com produções de IA, o que destaca a dificuldade de estabelecer uma fronteira clara entre código gerado por IA e humano apenas com base em critérios superficiais.

A conclusão a que se chega é que, embora os critérios ajudem no processo de análise, é necessária a inclusão de ferramentas mais robustas e métricas objetivas para complementar essa avaliação. A subjetividade envolvida na interpretação de aspectos como a clareza dos identificadores ou a lógica dos comentários pode induzir ao erro, e a padronização de práticas de programação assistida por IA precisa incorporar métodos mais consistentes e mensuráveis.

Portanto, os resultados sugerem que as diretrizes desenvolvidas para a programação assistida por IA devem não apenas reforçar a importância desses critérios, mas também propor o uso de ferramentas automáticas de análise de código, que possam identificar padrões específicos de IA ou humanos de forma mais precisa. Além disso, treinar programadores para reconhecer características mais sutis na codificação de IA pode ajudar a aumentar a precisão dessa distinção.

Critério	Síntese da Opinião dos Participantes	Síntese formada sobre o grau de utilidade para Identificar Código Gerado por IA ou Humano
Formatação	Os participantes notaram que códigos gerados por IA geralmente são bem formatados, mas podem ter falhas em consistência e uso de padrões como switch. A formatação humana tende a variar mais, dependendo da experiência do programador.	**Alta**. IA costuma gerar códigos mais padronizados e consistentes, enquanto humanos podem introduzir variações com base no estilo ou nível de experiência.
Número de linhas	O número de linhas foi mencionado como um indicativo de eficiência, com IA produzindo códigos mais enxutos, enquanto programadores humanos podem escrever de forma mais redundante.	**Moderada**. A concisão do código pode ser um indicativo, já que IA tende a otimizar o número de linhas, mas nem sempre é suficiente para diferenciar claramente.
Comentários	A falta ou presença de comentários foi um ponto forte. A IA frequentemente adiciona comentários padronizados, enquanto humanos podem ser inconsistentes ou omitir completamente.	**Alta**. Códigos gerados por IA geralmente incluem comentários superficiais ou genéricos, enquanto os humanos podem comentar de forma mais personalizada ou não comentar nada.
Átomos de Confusão	Muitos participantes relataram não encontrar átomos de confusão, ou encontrá-los de forma esparsa. A IA, em geral, minimiza erros lógicos ou sintáticos, enquanto humanos podem gerar mais confusão ao lidar com códigos complexos.	**Baixa**. Embora a IA reduza a presença de átomos de confusão, esse critério não foi suficiente para distinguir claramente entre código humano ou de IA.
Nome de Identificadores	IA tende a usar nomes descritivos e claros, enquanto humanos podem variar de acordo com sua própria lógica ou estilo. Muitos participantes observaram que os identificadores da IA são mais consistentes e organizados.	**Moderada**. Identificadores mais consistentes e descritivos são comuns em código gerado por IA, mas isso também pode ser feito por programadores experientes, tornando o critério útil, mas não conclusivo.

Tabela 2:Síntese de opiniões

A partir do contexto organizacional e da pesquisa em TI, o conhecimento tácito é reconhecido como o saber que se encontra embutido nas experiências pessoais, comportamentos e percepções dos indivíduos [47]. Esse tipo de conhecimento é subjetivo,

difícil de codificar e, muitas vezes, é comunicado apenas por meio da interação social [48]. No caso de profissionais de TI, o compartilhamento de conhecimento tácito depende de motivações individuais e culturais que vão além de métodos objetivos de avaliação [49].

Já o conhecimento explícito pode ser formalizado e compartilhado de forma mais objetiva e direta [47]. Nas avaliações como as do teste de Turing, a capacidade de explicitar critérios de julgamento e evidências pode ser vista como uma expressão desse tipo de conhecimento. Porém, nas observações do presente estudo, verificou-se que os avaliadores não conseguiram estabelecer todos os critérios de forma clara e explícita, sugerindo a presença de conhecimento tácito na própria avaliação.

A experiência pessoal foi citada como um critério por alguns participantes, mas seu uso é problemático. Sendo a experiência uma forma de conhecimento tácito, ela não pode ser completamente externalizada ou objetivamente comparada, o que a torna um critério abstrato. Esse tipo de avaliação subjetiva carece de um referencial explicitamente compartilhado entre avaliadores, o que dificulta a replicabilidade e a clareza dos critérios de avaliação. Na prática, isso aponta para a influência do conhecimento tácito nas decisões e avaliações, como nos testes de Turing, onde subjetividade e percepção individual afetam a consistência dos julgamentos.

4.8. Análise geral

Como pode ser visto na Figura 16, dada a baixa taxa de acerto na identificação da origem dos códigos, apenas 30% dos entrevistados acertaram com exatidão e o ideal seria que todos tivesse acertado, fica nítida a necessidade urgente de desenvolver soluções que aprimorem a compreensão e a análise dos códigos gerados por IA na o âmbito acadêmico dos cursos de TI. As recomendações práticas a seguir visam otimizar a eficiência e a qualidade na geração de código assistida por IA, além de abordar os impactos dessas práticas na indústria e na comunidade acadêmica.

Figura 16: Gráfico da quantidade de acertos

1. **Capacitação Contínua**: Investir em treinamentos regulares e workshops sobre programação assistida por IA para desenvolvedores e equipes de TI. Isso deve incluir não apenas o uso das ferramentas, mas também o entendimento das características dos códigos gerados por IA, permitindo que os profissionais desenvolvam uma percepção crítica sobre a autoria e a qualidade do código.
2. **Criação de Diretrizes de Qualidade**: Estabelecer um conjunto de diretrizes e melhores práticas para a revisão de código gerado por IA. Isso deve incluir critérios claros para avaliar a legibilidade, a clareza dos identificadores, a estrutura do código e a presença de comentários que ajudem a entender a lógica do programa.
3. **Ferramentas de Análise de Código**: Desenvolver e implementar ferramentas de análise que possam identificar automaticamente padrões que indiquem a autoria de IA, como a utilização excessiva de comentários ou a presença de redundâncias. Isso ajudará a automatizar a revisão de código e a aumentar a precisão na identificação de sua origem.
4. ***Feedback* Estruturado**: Criar um sistema de *feedback* onde programadores possam compartilhar suas experiências com códigos gerados por IA. Esse sistema poderia incluir uma plataforma para discutir os desafios e as armadilhas que surgem ao trabalhar com esses códigos, promovendo um aprendizado coletivo.
5. **Integração de AI e Revisão Humana**: Promover a colaboração entre IA e desenvolvedores humanos na geração de código. Isso pode ser feito mediante um sistema em que a IA sugira código, mas a revisão final e as decisões de

implementação fiquem a cargo de um humano. Essa abordagem pode garantir que a qualidade e a legibilidade sejam mantidas.

6. **Estudos de Caso e Pesquisa Acadêmica**: Incentivar estudos de caso e pesquisas na comunidade acadêmica sobre o impacto da programação assistida por IA na qualidade do código e na produtividade dos desenvolvedores. Isso pode ajudar a estabelecer uma base teórica sólida que guie futuras práticas na indústria.
7. **Comunicação entre Indústria e Academia**: Fomentar uma maior colaboração entre as indústrias de TI e as instituições acadêmicas para discutir as tendências em programação assistida por IA e os desafios emergentes. Essa interação pode facilitar a troca de conhecimentos e práticas que beneficiem ambos os lados.

Essas soluções visam não apenas melhorar a qualidade do código gerado por IA, mas também preparar os profissionais da indústria de TI para um futuro onde a programação assistida por IA se tornará cada vez mais prevalente. Ao adotar essas diretrizes, espera-se que a indústria se torne mais eficiente e capaz de enfrentar os desafios associados a essa nova era da programação.

5. CONCLUSÕES E TRABALHOS FUTUROS

O presente estudo explorou a capacidade da IA, representada por ChatGPT e Bard, de gerar código-fonte em Java que pode ser confundido com criações humanas. Os objetivos inicialmente traçados foram plenamente alcançados, evidenciando a necessidade de desenvolver soluções que melhorem a compreensão e análise dos códigos gerados pela IA no contexto acadêmico dos cursos de informática.

Os resultados obtidos mostraram que a maioria dos futuros profissionais de TI não sabe identificar corretamente se um código foi gerado por uma inteligência artificial ou não, já que 30% das pessoas adivinharam a origem dos 3 códigos. A legibilidade, consistência, semântica e estrutura lógica dos códigos gerados por IA são notavelmente semelhantes aos códigos escritos por humanos, com 40% das pessoas entendendo mal a origem do código gerado pelo ChatGPT e 50% dizendo que não entendem a origem do código gerado pelo Bard, questionando a distinção entre autoria de códigos-fonte.

O estudo revelou informações valiosas sobre a crescente capacidade da inteligência artificial de replicar elementos do raciocínio humano na geração de código-fonte. Com a capacidade de ferramentas como ChatGPT e Bard de produzir código Java que se assemelha ao trabalho humano, tornou-se claro que a percepção de autoria entre IA e programadores humanos está se tornando cada vez mais difícil de distinguir. A falta de clareza entre o que a IA gera e o que os programadores criam pode ter implicações profundas para a educação em programação, onde a compreensão da autoria e construção do código se torna crítica para a aprendizagem.

Além disso, o trabalho destaca a importância de soluções que ajudem os futuros profissionais de TI a desenvolver habilidades para distinguir entre produções humanas e artificiais, tendo em vista que, nos testes aplicados, uma parte significativa dos alunos teve dificuldade em discernir a origem dos códigos analisados. Isso mostra que, mesmo com o avanço das ferramentas de inteligência artificial, a formação técnica e crítica dos futuros profissionais deve se adequar ao contexto tecnológico atual.

As descobertas sugerem que, à medida que a IA evolui, a sua contribuição e influência na criação de software podem tornar-se menos visíveis, mas mais profundas, e talvez inevitáveis, na programação quotidiana. Com isso, o estudo sugere que, no futuro, a distinção entre "criador" e "assistente inteligente" no contexto da autoria de código-fonte poderá exigir novas abordagens de avaliação e análise, não apenas no contexto educacional, mas também nas indústrias de TI e nas regulamentações tecnológicas. Em última análise, os resultados obtidos destacam a necessidade de debates futuros sobre o impacto ético e educacional da IA na autoria e originalidade do código, abrindo caminho para uma nova era em que a inteligência artificial e os humanos trabalhem de forma colaborativa e mais integrada no desenvolvimento de software.

5.1. Limitações e Problemas Identificados

Contudo, algumas limitações e desafios surgiram durante o desenvolvimento da pesquisa. Uma limitação notável foi a necessidade de um grupo mais diversificado de participantes, com maior representação em termos de experiência e conhecimento de programação. Além disso, a falta de ferramentas padronizadas para avaliar a qualidade e, secundariamente, a autoria dos códigos fez com que o processo de análise ficasse mais dependente do julgamento pessoal dos participantes.

5.2. Trabalhos Futuros

As descobertas deste estudo abrem portas para diversas oportunidades e trabalhos futuros. Alguns dos caminhos a serem explorados incluem:

- Ampliação da Amostra de Avaliadores: realizar experimentos com um grupo mais diversificado de avaliadores, incluindo programadores com diferentes níveis de experiência, a fim de obter uma compreensão mais abrangente da percepção humana na diferenciação entre códigos gerados por IA e humanos.
- Abranger mais critérios: isso envolveria mais análises qualitativas das respostas para identificar as métricas mais relevantes na distinção entre as origens dos códigos.
- Análise Comparativa de Diferentes IAs: um estudo futuro pode se focar na comparação de várias outras IA's para geração de código, avaliando como cada uma trata problemas de redundância, otimização, comentários e legibilidade.

- Análise detalhada de conhecimento tácito: trabalhos futuros devem discutir os níveis de conhecimento tácito e explícito com mais profundidade, visto que as pessoas entrevistadas neste trabalho não conseguiram citar todos os critérios explícitos que elas usaram na avaliação, recorrendo ao uso de critérios tácitos.

Esses trabalhos futuros oferecem oportunidades para expandir os conhecimentos adquiridos neste estudo, abrindo novas perspectivas para pesquisas e aplicações práticas das IA na geração de códigos-fonte.

5.3. Considerações Finais

O estudo trouxe contribuições significativas, demonstrando a dificuldade que os futuros profissionais da área de TI têm em identificar a origem dos códigos e que já é fato a necessidade de correção desse erro [45]. Embora apresente limitações, as descobertas desta pesquisa fornecem uma base para futuras explorações no campo da Inteligência Artificial, sinalizando possíveis direções para a evolução dessas tecnologias e correção na competência dos futuros profissionais. As oportunidades de ampliação e aprimoramento apresentadas reforçam a relevância deste estudo no contexto em constante evolução da Tecnologia da Informação e Inteligência Artificial.

REFERÊNCIAS

[1] TURING, A.M. MIND A QUARTERLY REVIEW OF PSYCHOLOGY AND PHILOSOPHY: I.—COMPUTING MACHINERY AND INTELLIGENCE. **The Mind Association**, v.LIX, n.236, p.433–460, Outubro, 1950.

[2] OMAN, Paul W.; COOK, Curtis R. Typographic style is more than cosmetic. **Communications of the ACM**, v.33, n.5, p.506–520, Maio, 1990.

[3] MI, Quing; ZHAN, Yi; WENG, Han; BAO, Quinghang; CUI, Longjie; MA, Wei. A graph-based code representation method to improve code readability classification. **Empirical Software Engineering**, v.28, n.4, p.26, Maio, 2023.

[4] COSTA, José Aldo; GHEYI, Rohit; CASTOR, Fernando; OLIVEIRA, Pablo; RIBEIRO, Márcio; FONSECA, Baldoino. Seeing confusion through a new lens: on the impact of atoms of confusion on novices' code comprehension. **Empirical Software Engineering**, v.28, n.4, p.48, Maio, 2023.

[5] SANTOS, Reydne Bruno Dos. **Um estudo sobre definição e avaliação da readability e legibility do código fonte**. 2021. Dissertação – Centro de Informática, Universidade Federal de Pernambuco-UFPE, Recife, 2021.

[6] MIRANDA, Túlio José Prestrelo. Código 1. João Pessoa: GitHub, Inc., 2023. Disponível em: https://github.com/tulio700/codigo_1. Acesso em: 8, Nov, 2023.

[7] MIRANDA, Túlio José Prestrelo. Código 2. João Pessoa: GitHub, Inc., 2023. Disponível em: https://github.com/tulio700/codigo_2. Acesso em: 8, Nov, 2023.

[8] MIRANDA, Túlio José Prestrelo. Código 3. João Pessoa: GitHub, Inc., 2023. Disponível em: https://github.com/tulio700/codigo_3. Acesso em: 8, Nov, 2023.

[9] BUSE, Raymond P.L.; WEIMER, Westley R. Learning a Metric for Code Readability. **IEEE Transactions on Software Engineering**. v.36, n4, p.546 - 558, Novembro, 2009.

[10] LEE, Taek; LEE, Jung; IN, Hoh Peter. A study of different coding styles affecting code readability. **International Journal of Software Engineering and its Applications**. v7, p.413-422, Setembro, 2013.

[11] YU K-H, BEAM AL and KOHANE IS. Artificial Intelligence in healthcare. **Nat Biomed Eng**. v.2, n.10, p: 719–731, 2018.

[12] SHAHRIAR, Sabik. GAN computers generate arts? A survey on visual arts, music, and literary text generation using generative adversarial network. **Science Direct**, 73: 102237, 2022

[13] RADFORD, Alec; WU, Jeffrey; CHILD, Rewon; et al. Language models are unsupervised multitask learners. **OpenAI Blog**, v.1,n.8: p.9, 2019.

[14] SHAHRIAR, Sabik; HAYAWI, Kadhim. Let's have a chat! A conversation with ChatGPT: technology, applications, and limitations. Bon View; n.1:p.4, Julho, 2023.

[15] GEORGE A. Shaji; GEORGE A. S. Hovan. A review of ChatGPT AI's impact on several business sectors. **Partner Univers Int Innov**, v.1,n.1:p. 9–23, Janeiro, 2023.

[16] BANG, Yejin; CAHYAWIJAYA, Samuel; LEE, Nayeon; DAI, Wenliang; et al. A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. **arXiv**, v.4, n.1, Novembro, 2023.

[17] BEUTEL, Gernot; GEERITS Eline; KIELSTEIN, Jan T.. Artificial hallucination: GPT on LSD? **Crit Care**, v.27, n.1: p.148, Maio, 2023.

[18] MANAKUL, Potsawee; LIUSIE, Adian; GALES, Mark J. F.. SelfCheckGPT: zero-resource black-box hallucination detection for generative large language models. **arXiv**, v.3, n.1, Outubro, 2023.

[19] JI, Jiaming ; LIU, Mickel; DAI, Juntao; PAN, Xuehai; et al. BeaverTails: towards improved safety alignment of LLM via a human-preference dataset. **arXiv**, v.3, n.1, Novembro 2023.

[20] DWIVEDI, Yogesh K.; KSHETRI, Nir; HUGHES, Laurie; SLADE, Emma Louise; et al. Opinion paper: 'So what if ChatGPT wrote it?' Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. **International Journal Information Management**, v.71, Agosto, 2023.

[21] HARRER, Stefan. Attention is not all you need: the complicated case of ethically using large language models in healthcare and medicine. **EBioMedicine**; v.90: 104512, Março, 2023.

[22] JUNGHERR, Andrea. Using ChatGPT and other large language model (LLM) applications for academic paper assignments. **SocArXiv**, v.1, Março, 2023.

[23] KHALIL Mohammad; Er, Erkan. Will ChatGPT get you caught? Rethinking of plagiarism detection. **arXiv**, v.1, Fevereiro, 2023.

[24] COTTON, Debby R.E.; COTTON, Peter A. ; SHIPWAY, J. Reuben;. Chatting and cheating: ensuring academic integrity in the era of ChatGPT. **SocArXiv**, v.1 Janeiro, 2023.

[25] PERKINS Mike. Academic Integrity considerations of AI large language models in the post-pandemic era: ChatGPT and beyond. **Journal of University Teaching and Learning Practice**, v.20, n.2: p.7, Janeiro, 2023.

[26] SCHACH, Stephen R. **Object-oriented and Classical Software Engineering**. Nova York: Mcgraw Hill Higher Education, 2006.

[27] AHO, Alfred; ULLMAN, Jeffrey; SETHI, Ravi; LAM, Monica. **Compilers: Principles, Techniques, and Tools**. Boston: Addison Wesley, 2006.

[28] BLOCH, Joshua. **Effective Java**. Boston: Addison Wesley, 2017.

[29] HORSTMANN, Cay. **Core Java: Fundamentals, Volume 1**. Austin: Oracle Press, 2021.

[30] ELLISON, Larry; OATES, Ed; MINER, Bob. Java Documentation. Santa Clara: Oracle, Inc., 1978. Disponível em: https://docs.oracle.com/en/java/. Acesso em: 13, Nov, 2024.

[31] VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; et al. Attention Is All You Need, **arXiv**, v.7, Agosto, 2023.

[32] ALTMAN, Sam; HODDMAN, Reid; LIVINGSTON, Jessica; MUSK, Elon; et al. OpenAI. São Francisco: OpenAI, Inc., 2015. Disponível em: https://openai.com/research/. Acesso em: 13, Nov, 2024.

[33] TEAM, Gemini; ANIL, Rohan; BORGEAUD, Sebastian; WU, Yonghui; et al.Gemini: A Family of Highly Capable Multimodal Models, **Hugging Face**, v.1, n.1, Dezembro, 2023.

[34] PICHAI, Sundar; HASSABIS, Demis. Introducing Gemini: our largest and most capable AI model, **India Blog**, v.1, n.1, Dezembro, 2023.

[35] MEIRELLES, Paulo Roberto Miranda, **Monitoramento de métricas de código-fonte em projetos de software livre**. 2013. Tese - Instituto de Matemática e Estatística da Universidade de São Paulo. São Paulo. 2013.

[36] FOWLER, Martin, **Refactoring: Improving the Design of Existing Code**. Boston: Addison-Wesley, 2018.

[37] GOPSTEIN, Dan; IANNACONE, Jake; YAN, Yu; DELONG, Lois; ZHUANG, Yanyan; YEHMK-C, Martin; CAPPOS, Justin. Understanding Misunderstandings in Source Code. In: Proceedings of the Joint Meeting on Foundations of Software Engineering, ESEC/FSE'17. Entidade patrocinadora do evento: Association for Computing Machinery, New York, p. 129–139, Setembro, 2017.

[38] GOPSTEIN, Dan; ZHOU, Hongwei, Henry; FRANKL, Phyllis; CAPPOS, Justin. Prevalence of Confusing Code in Software Projects:Atoms of Confusion in theWild. In: Proceedings of the International Conference on Mining Software Repositories, ICSMR'18. Entidade patrocinadora do evento: Association for Computing Machinery, New York, p. 281–291, Maio, 2018.

[39] MARTIN, Robert C. **Clean Code: A Handbook of Agile Software Craftsmanship**. Londres: Pearson, 2008.

[40] MIRANDA, Túlio José Prestrelo. Código 1. João Pessoa: GitHub, Inc., 2023. Disponível em: https://docs.google.com/forms/d/e/1FAIpQLScErfxL46O7IS_lWQlbiB74ljG1BVw4OGpSrOboZ1b7LygO1g/viewform?usp=sf_link. Acesso em: 22, Out, 2024.

[41] MCCONNELL, Steve. **Code Complete: A Practical Handbook of Software Construction, Second Edition**. Redmond: Microsoft Press, 2004.

[42] BROWN, Tom B.; MANN, Benjamin; RYDER, Nick; SUBBIAH, Melanie; et al. Language Models are Few-Shot Learners. **arXiv**,vol. 4, no. 14165, pp. 1-75, Jul, 2020

[43] MCCABE, T.J. A Complexity Measure. **IEEE Transactions on Software Engineering** , vol. SE-2, no. 4, pP..308-320, Dec, 1976.

[44] NAGAPPAN, Nachiappan; BALL, Thomas; ZELLER, Andreas. Mining metrics to predict component failures. In: International Conference On Software Engineering, 28th, Shanghai. **ISBN: 1595933751**; **DOI: 10.1145/1134285** Entidade patrocinadora do evento: Association for Computing Machinery(ACM); ACM Special Interest Group on Software Engineering(SIGSOFT), 2006.

[45] HAYAWI, Kadhim; SHAHRIAR, Sakib; MATHEW, Sujith S. The imitation game: Detecting human and AI-generated texts in the era of ChatGPT and BARD. **Journal of Information Science**, vol. 1, no. 1, pp. 1-36, Fev, 2024.

[46] LANGHOUT, Chris; ANICHE Maurício. Atoms of Confusion in Java. In: Proceedings of the International Conference on Program Comprehension. **IEEE**, ICPC'21, pp 25–35, Maio, 2021.

[47] BORGES, Renata. Tacit knowledge sharing between IT workers: The role of organizational culture, personality, and social environment. **Emerald**, vol. 36, n. 1: p. 89 - 108, 2013.

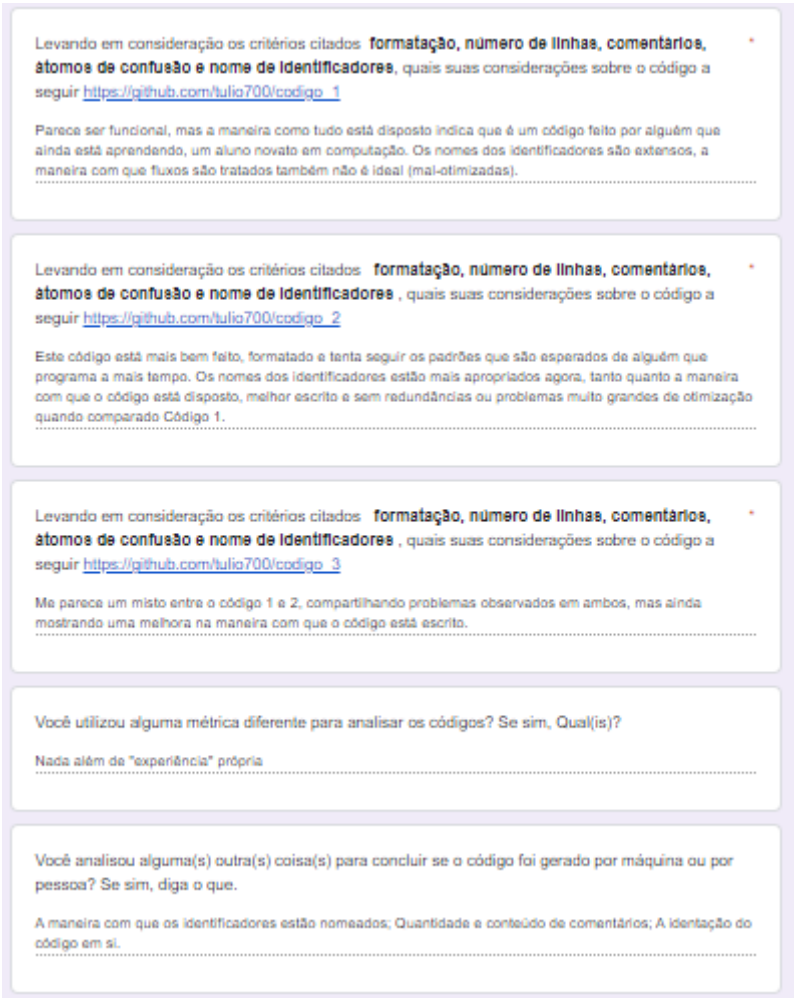
[48] ALWIS, Ragna Seidler-de; HARTMANN, Evi; GEMÜNDEN, Hans Georg. The role of tacit knowledge in innovation management. Annual IMP Conference in Copenhagen, 20th, Janeiro, 2004.

[49] NONAKA, Ikujiro; KROGH, Georg von. Tacit Knowledge and Knowledge Conversion: Controversy and Advancement in Organizational Knowledge Creation Theory. **Perspective**, vol. 20, no. 3: p. 635 - 652, Maio, 2009.

[50] SERGEYUK, Agnia; LVOVA, Olga; TITOV, Sergey; SEROVA, Anastasiia; et al. Reassessing Java Code Readability Models with a Human-Centered Approach. In: The 32nd IEEE/ACM International Conference on Program Comprehension, Lisbon, Portugal, Abril, 2024.

APÊNDICES

Perguntas e respostas do questionário para coleta e análise de dados.



Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Parece ser funcional, mas a maneira como tudo está disposto indica que é um código feito por alguém que ainda está aprendendo, um aluno novato em computação. Os nomes dos identificadores são extensos, a maneira com que fluxos são tratados também não é ideal (mal-otimizadas).

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Este código está mais bem feito, formatado e tenta seguir os padrões que são esperados de alguém que programa a mais tempo. Os nomes dos identificadores estão mais apropriados agora, tanto quanto a maneira com que o código está disposto, melhor escrito e sem redundâncias ou problemas muito grandes de otimização quando comparado Código 1.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Me parece um misto entre o código 1 e 2, compartilhando problemas observados em ambos, mas ainda mostrando uma melhora na maneira com que o código está escrito.

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Nada além de "experiência" própria

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

A maneira com que os identificadores estão nomeados; Quantidade e conteúdo de comentários; A identação do código em si.

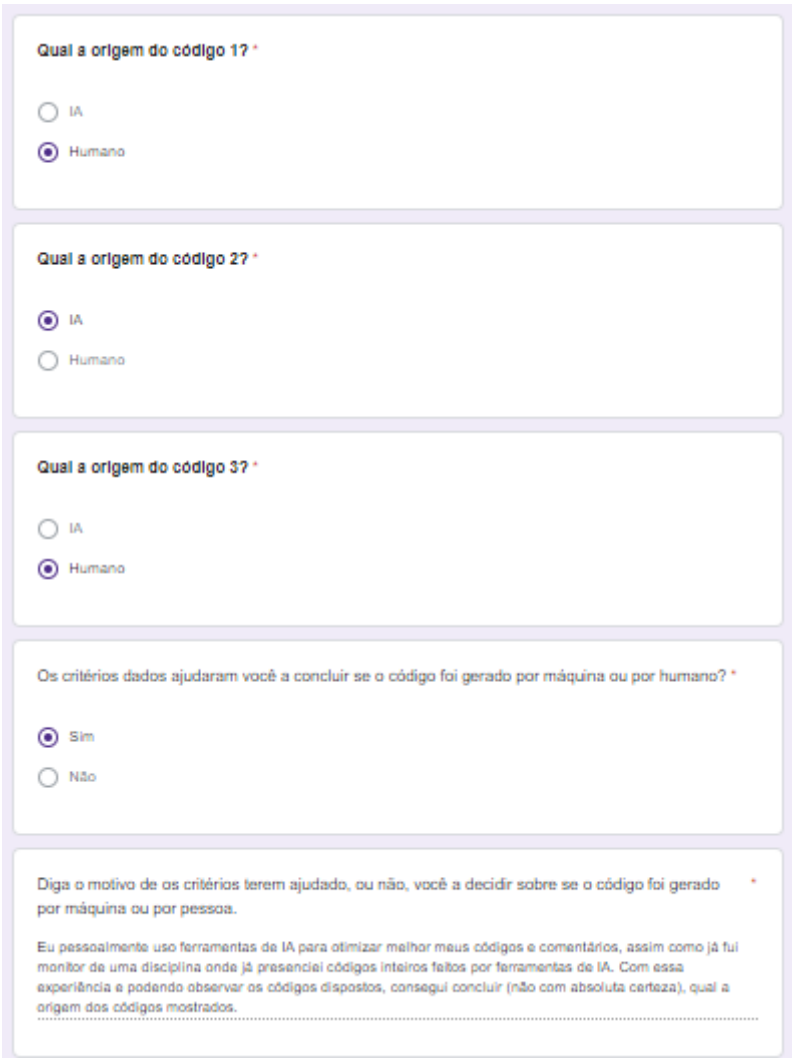


Figura 16: Formulário aluno 1

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Número de linhas muito grande e código confuso

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Apenas algumas partes do código estão comentando, geralmente IA comenta todas as funções

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Um código menor e que dá pra entender bem

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

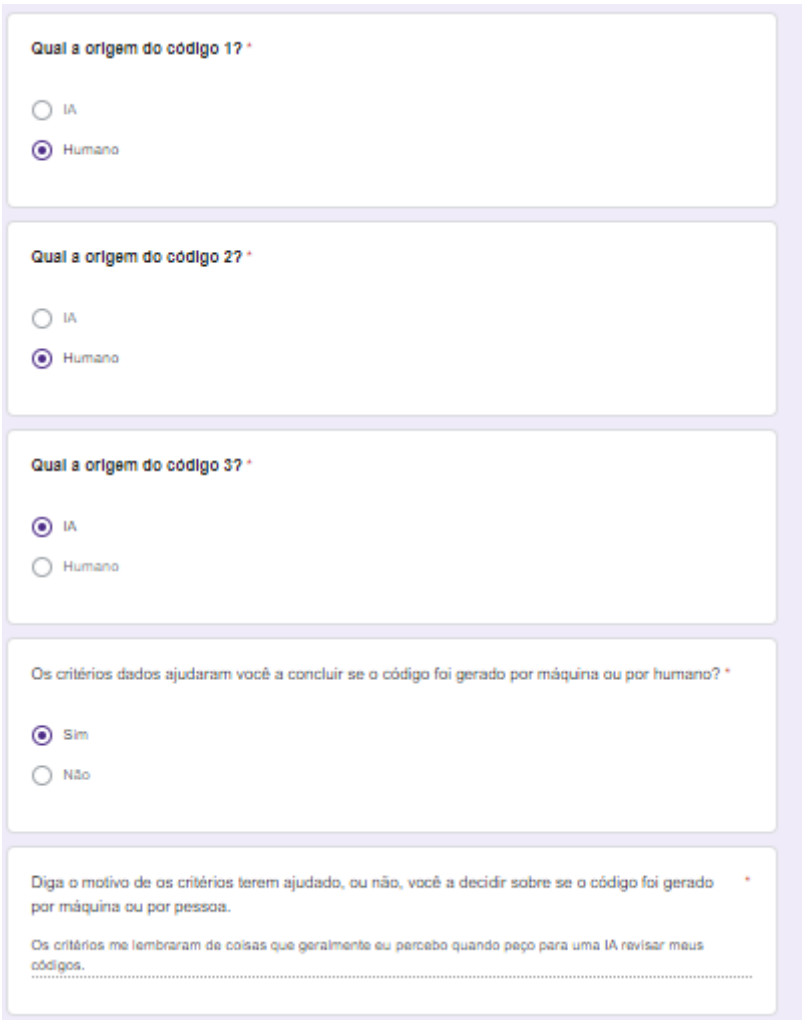


Figura 17: Formulário aluno 2

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Formatação ruim, poucos espaços entre blocos de código e muitos blocos condicionais aninhados, fica pouco legível o código.

Código ficou longo, mas não acho algo necessariamente ruim para esse exemplo.

Não possui comentários.

Não identifiquei átomos de confusão, mas eu já precisei programar um Léxico e por isso sei o que o código está fazendo. Se eu não tivesse conhecimento prévio, provavelmente estaria confuso.

Nomes dos identificadores estão bem claro.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Achei bem formatado e legível o código.

Tamanho bom.

Não possui comentários.

Não identifiquei átomos de confusão.

Nomes dos identificadores estão bem claros.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Achei bem formatado e legível o código.

Tamanho bom.

Não possui comentários.

Não identifiquei átomos de confusão.

Nomes dos identificadores estão bem claros.

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

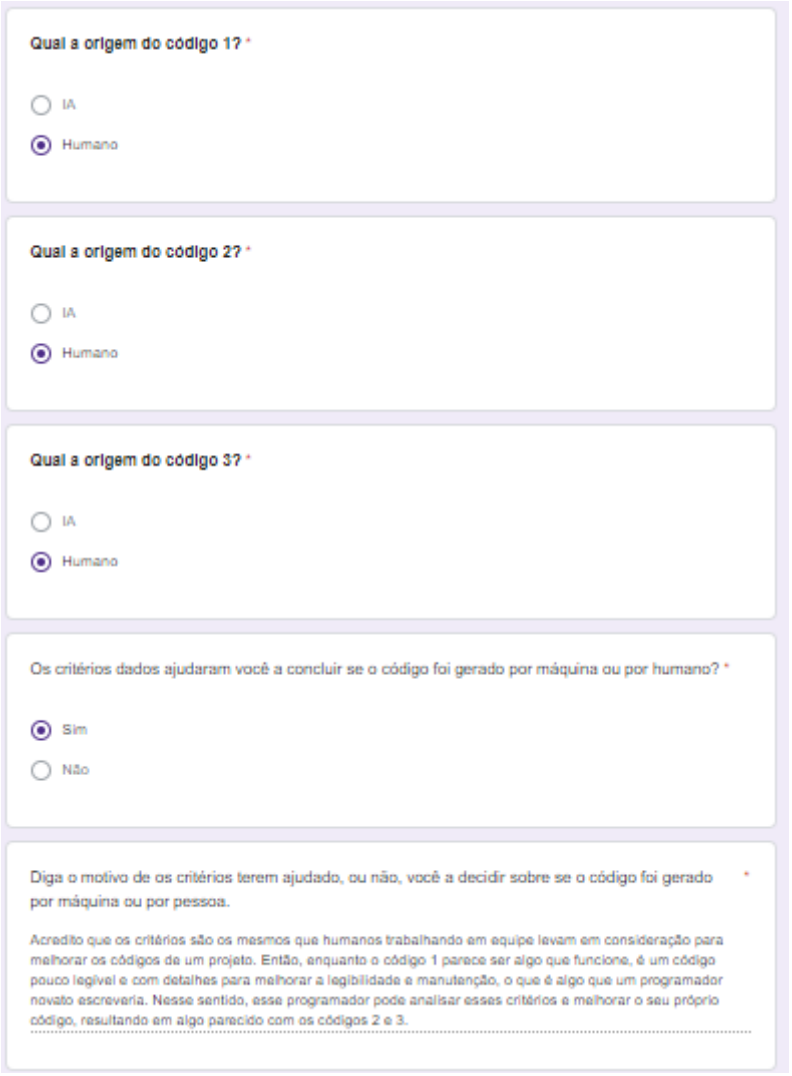


Figura 18: Formulário aluno 3

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

o código parece estar formatado corretamente, mas os nomes das funções não deixam muito claro qual a sua funcionalidade, além disso os ifs aninhados deixam a legibilidade terrível

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

esse não tem o problema de ifs aninhados e tem funções com nomes mais claros, entretando alguns valores boleanos utilizados em laços while e if ainda são muito grandes e difíceis de ler

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

esse é o melhor dos 3, os ifs repetitivos foram substituídos por um switch, o que deixa muito mais legível, além disso os booleanos longos foram quebrados em algumas linhas, com funções com nomes mais claros. O terceiro não parece ter sido produzido por ia por ter uma lógica um pouco mais complexa.

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

sim, geralmente o código da ia tem uma qualidade mediana, não é ruim, mas não chega a ser ideal, além disso normalmente as principais ias costumam adicionar comentários em algumas partes específicas do código

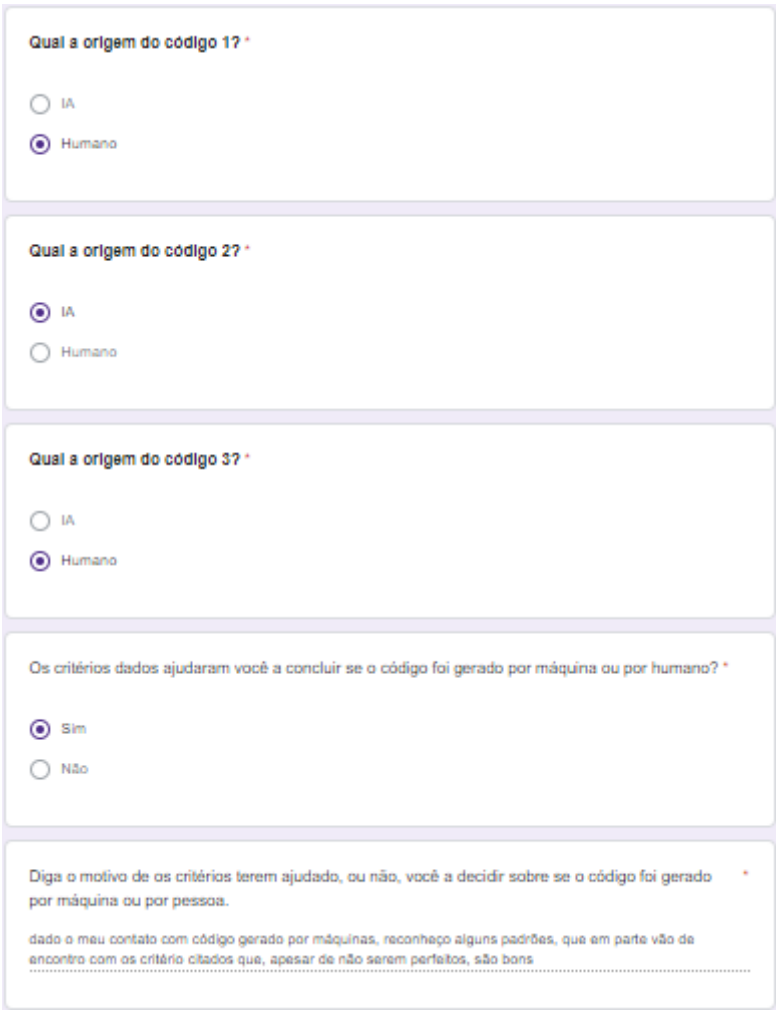


Figura 19: Formulário aluno 4

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

O código está bem indentado, tem um número adequadro de linhas para um analisador léxico (é o que imagino que seja), não tem comentários no código, quanto aos átomos de confusão, eu particularmente não vejo um código em Java há muito tempo, então eu me confundi em alguns momentos, e os nomes de identificadores estão bons e atendem às boas práticas de Java.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

O código parece mais complexo que o primeiro, mas manteve a boa indentação, diminuiu bastante os números de linha, os comentários no código demonstram coisas inacabadas, os átomos de confusão aumentaram, pois é um código de Java mais complexo que o anterior e eu teria que tomar bastante tempo para entender o que se passa e os identificadores ainda estão seguindo as boas práticas do Java.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

O código parece menos complexo que o segundo, mais complexo que o primeiro, mas manteve a boa indentação, tem um pouco mais de linhas do que o segundo, mas muito menos linhas que o primeiro, não tem comentários no código, os átomos de confusão se mantiveram em comparação ao segundo, pois é um código de Java mais complexo que o primeiro código e eu teria que tomar bastante tempo para entender o que se passa e os identificadores ainda estão seguindo as boas práticas do Java.

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Apenas o meu conhecimento de programação, compiladores e Java.

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

O primeiro código é muito redundante e, por experiência própria (dependendo da IA, principalmente se for ChatGPT), uma IA costuma cometer esse erro se o prompt de entrada não for muito preciso. O segundo código foi feito por um humano, principalmente porque o código parece mais "orgânico" (não sei dizer o porquê), e os comentários em condições inacabadas/sem comandos mostram que provavelmente foi um humano que fez aquele código. O terceiro código me deixa intrigada, pois usa apenas um módulo (o primeiro não usa nenhum e o segundo usa quatro), não tem comentários, e não é redundante como o primeiro, vou dizer que foi um ser humano que gerou, mas só porque não parei para olhar com muitos detalhes.

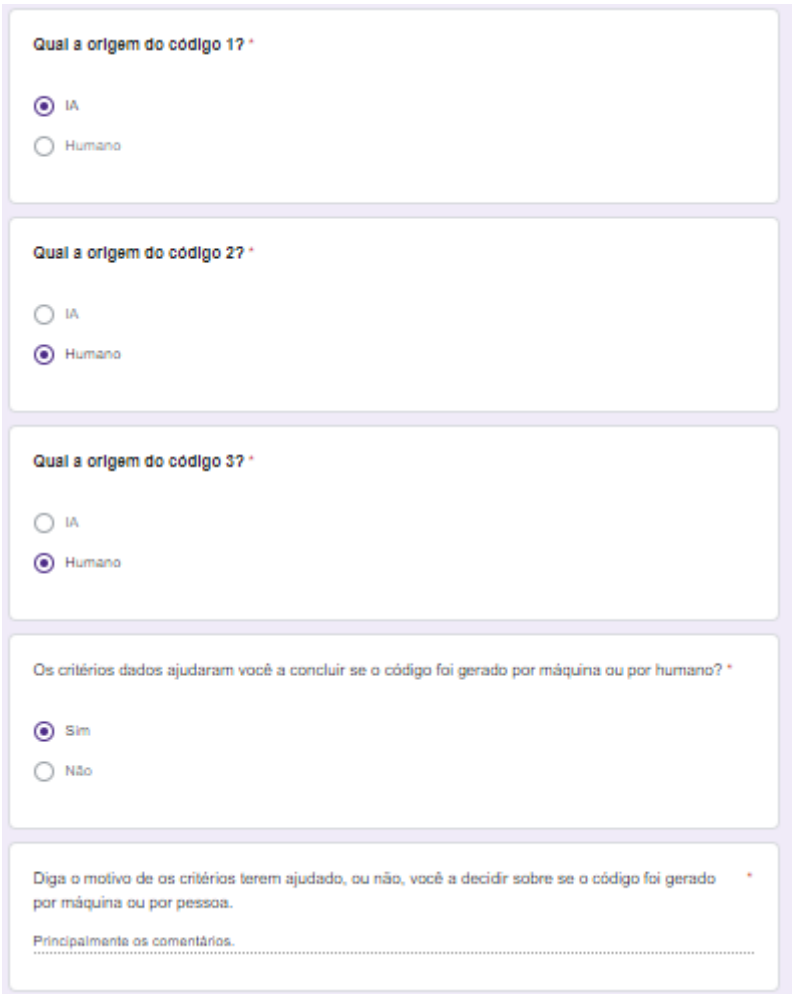


Figura 20: Formulário aluno 5

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Código robusto, mas bem indentado e legível ,porém sem comentários,oq dificulta o entendimento

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Mesmo do primeiro

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Mesmo do primeiro

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Não

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

Não

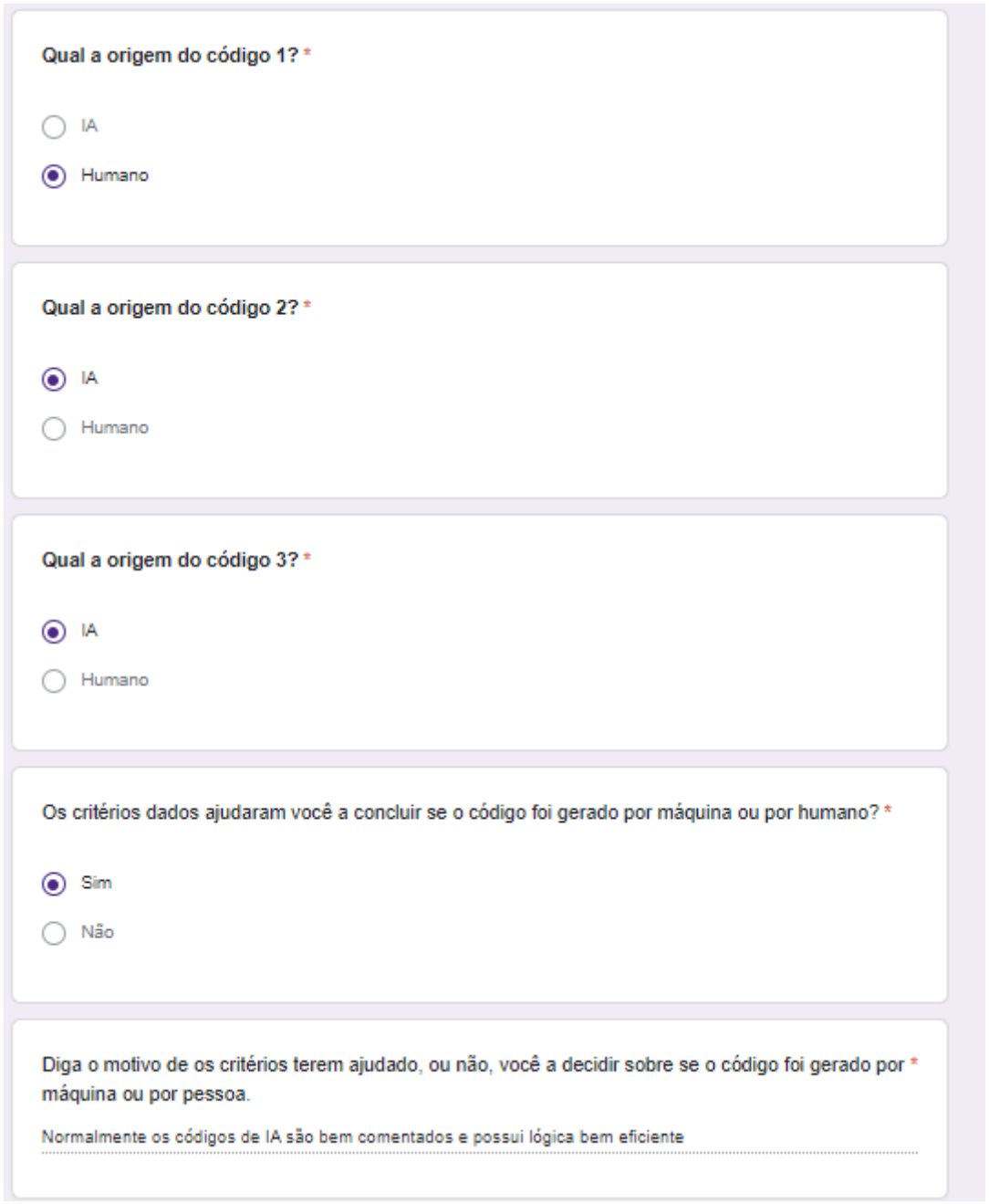


Figura 21: Formulário aluno 6

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

(Só olhei por cima mesmo dos códigos.) Acredito que poderia ser mais enxuto.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Cheiro de IA.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

ligeiramente suspeito

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

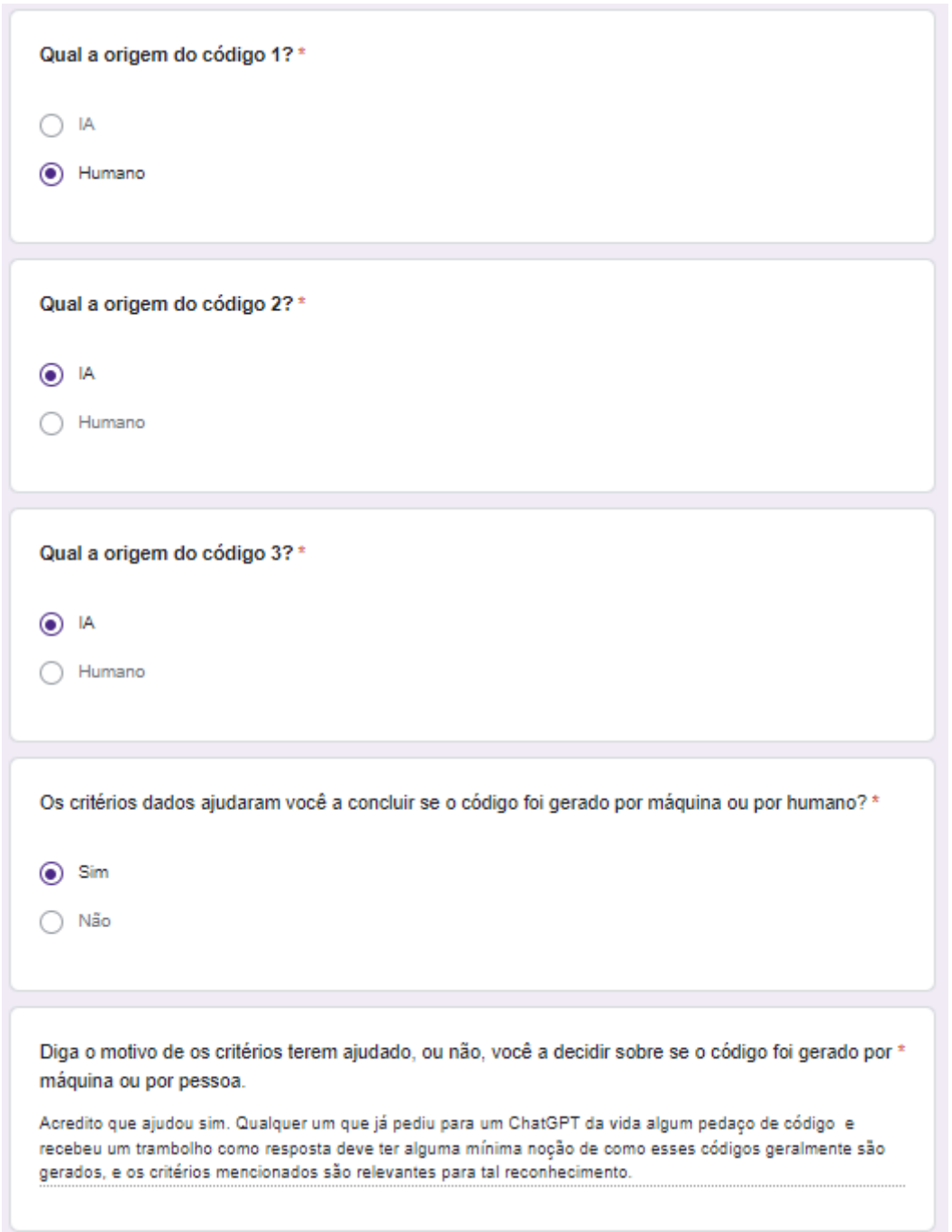


Figura 22: Formulário aluno 7

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Foi feito por IA

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Foi feito por humano

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Tive dúvidas mas acredito que tenha sido feito por IA

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

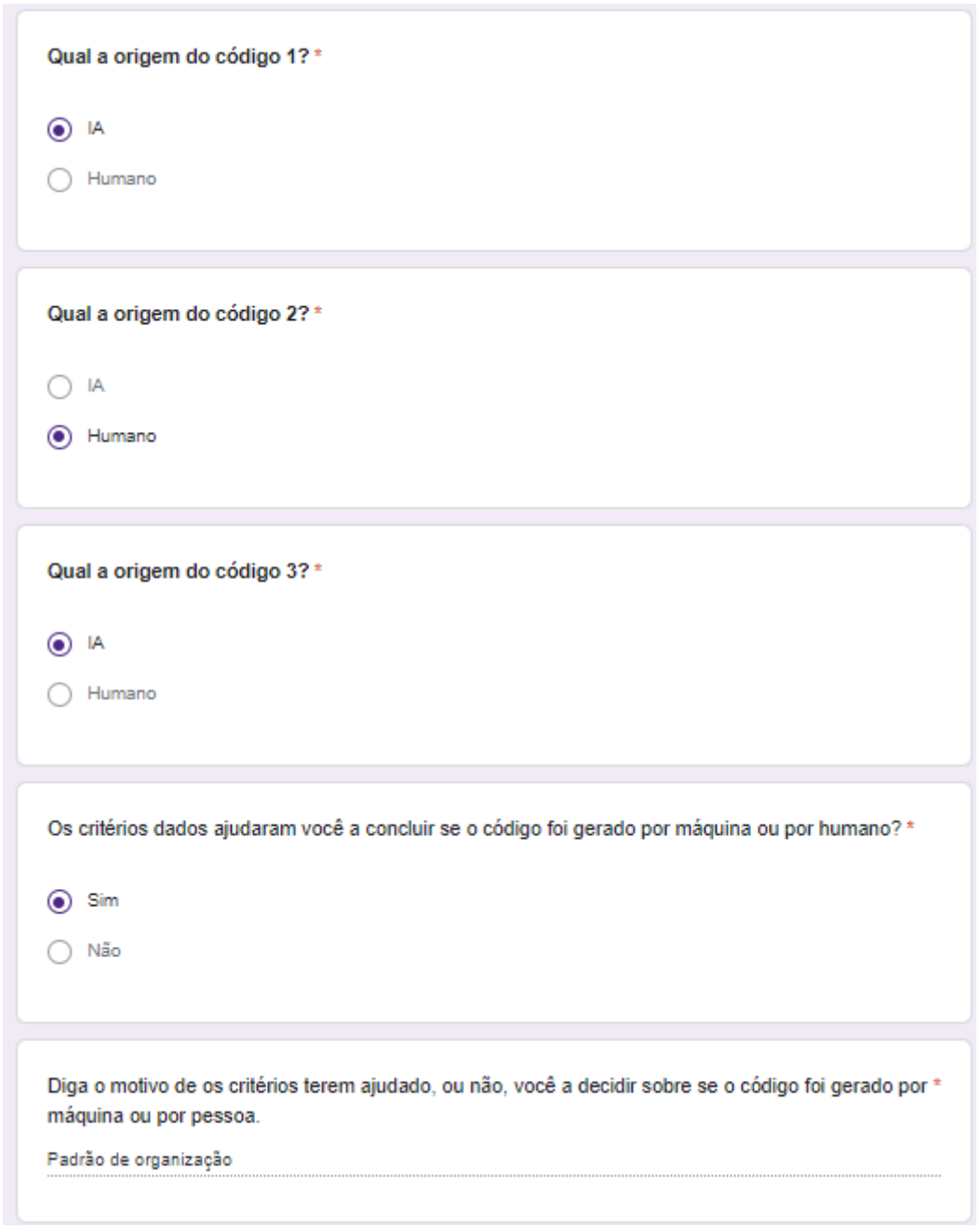


Figura 23: Formulário aluno 8

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Essa código, diferente dos outros, foi feito em Português. Tirando essa observação, não reparei em mais nada diferente. Pela simplicidade no nome da classe (Lexico), diria que foi gerado por humano.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Feito em Inglês. Em um dos elses (linha 72), o seu conteúdo não foi preenchido. Provavelmente, gerado por IA.

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Feito em Inglês.Não observei nada demais. Deve ter sido gerado por IA.

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

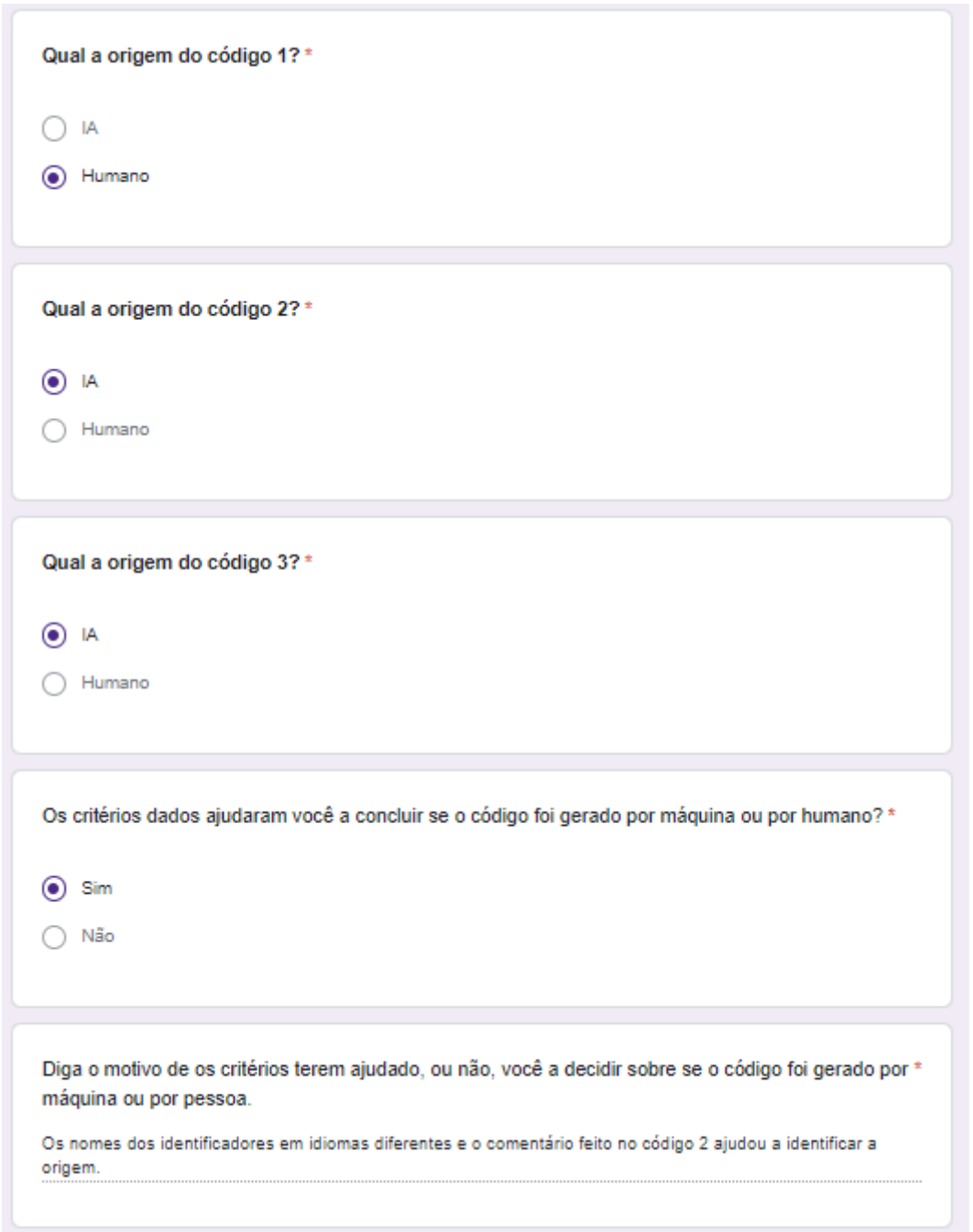


Figura 24: Formulário aluno 9

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores**, quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_1 *

Formatação: Codigo está bem estruturado e padronizado
Número de linhas: O código está bem extenso
Comentários: O código não possui nenhum
Átomos de confusão: Consegui entender a tokentização
Identificadores: Ou são bastante abreviados ou são bastante verbosos ou uma mistura dos dois

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_2 *

Formatação: Codigo está bem estruturado e padronizado
Número de linhas: O código está bem compacto
Comentários: Possui comentários de uma linha
Átomos de confusão: Achei um pouco confuso
Identificadores: São simples e explicativos

Levando em consideração os critérios citados **formatação, número de linhas, comentários, átomos de confusão e nome de identificadores** , quais suas considerações sobre o código a seguir https://github.com/tulio700/codigo_3 *

Formatação: Poderia ter uma função apenas para o swirtch case
Número de linhas: O código está bem compacto
Comentários: O código não possui nenhum
Átomos de confusão: Consegui entender a tokentização
Identificadores:

Você utilizou alguma métrica diferente para analisar os códigos? Se sim, Qual(is)?

Não

Você analisou alguma(s) outra(s) coisa(s) para concluir se o código foi gerado por máquina ou por pessoa? Se sim, diga o que.

Sim, se há importação de bibliotecas não nativas.

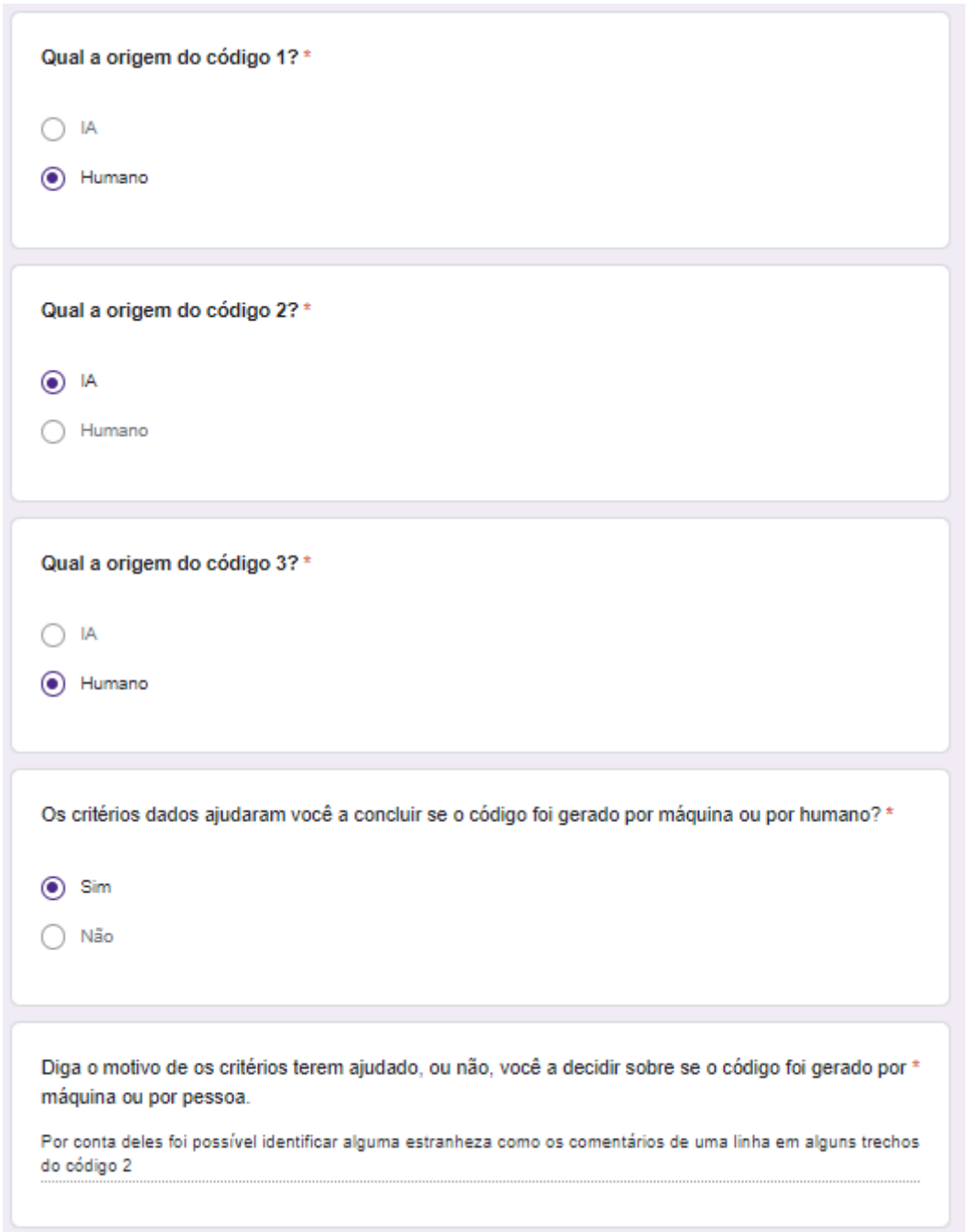


Figura 25: Formulário aluno 10

Códigos fonte utilizados para a pesquisa

```java
package lexico;

public class Lexico {

    LeitorDeArquivosTexto ldat;

    public Lexico(String arquivo) {
        ldat = new LeitorDeArquivosTexto(arquivo);
    }

    public Token proximoToken() {
        Token proximo = null;
        espacosEComentarios();
        ldat.confirmar();
        proximo = fim();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = palavrasChave();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = variavel();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = numeros();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = operadorAritmetico();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = operadorRelacional();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = delimitador();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = parenteses();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        proximo = cadeia();
        if (proximo == null) {
            ldat.zerar();
        } else {
            ldat.confirmar();
            return proximo;
        }
        System.err.println("Erro léxico!");
        System.err.println(ldat.toString());
        return null;
```

```
    }

    private Token operadorAritmetico() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == '*') {
            return new Token(TipoToken.OpAritMult, ldat.getLexema());
        } else if (c == '/') {
            return new Token(TipoToken.OpAritDiv, ldat.getLexema());
        } else if (c == '+') {
            return new Token(TipoToken.OpAritSoma, ldat.getLexema());
        } else if (c == '-') {
            return new Token(TipoToken.OpAritSub, ldat.getLexema());
        } else {
            return null;
        }
    }

    private Token delimitador() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == ';') {
            return new Token(TipoToken.Delim, ldat.getLexema());
        } else {
            return null;
        }
    }

    private Token parenteses() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == '(') {
            return new Token(TipoToken.AbrePar, ldat.getLexema());
        } else if (c == ')') {
            return new Token(TipoToken.FechaPar, ldat.getLexema());
        } else {
            return null;
        }
    }
```

```

    private Token operadorRelacional() {
        int caractereLido = ldat.lerProximoCaractere();
        char c = (char) caractereLido;
        if (c == '<') {
            c = (char) ldat.lerProximoCaractere();
            if (c == '>') {
                return new Token(TipoToken.OpRelDif, ldat.getLexema());
            } else if (c == '=') {
                return new Token(TipoToken.OpRelMenorIgual, ldat.getLexema());
            } else {
                ldat.retroceder();
                return new Token(TipoToken.OpRelMenor, ldat.getLexema());
            }
        } else if (c == '=') {
            return new Token(TipoToken.OpRelIgual, ldat.getLexema());
        } else if (c == '>') {
            c = (char) ldat.lerProximoCaractere();
            if (c == '=') {
                return new Token(TipoToken.OpRelMaiorIgual, ldat.getLexema());
            } else {
                ldat.retroceder();
                return new Token(TipoToken.OpRelMaior, ldat.getLexema());
            }
        }
        return null;
    }

    private Token numeros() {
        int estado = 1;
        while (true) {
            char c = (char) ldat.lerProximoCaractere();
            if (estado == 1) {
                if (Character.isDigit(c)) {
                    estado = 2;
                } else {
                    return null;
                }
            } else if (estado == 2) {
                if (c == '.') {
```

```
                c = (char) ldat.lerProximoCaractere();
                if (Character.isDigit(c)) {
                    estado = 3;
                } else {
                    return null;
                }
            } else if (!Character.isDigit(c)) {
                ldat.retroceder();
                return new Token(TipoToken.NumInt, ldat.getLexema());
            }
        } else if (estado == 3) {
            if (!Character.isDigit(c)) {
                ldat.retroceder();
                return new Token(TipoToken.NumReal, ldat.getLexema());
            }
        }
    }
}

private Token variavel() {
    int estado = 1;
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (estado == 1) {
            if (Character.isLetter(c)) {
                estado = 2;
            } else {
                return null;
            }
        } else if (estado == 2) {
            if (!Character.isLetterOrDigit(c)) {
                ldat.retroceder();
                return new Token(TipoToken.Var, ldat.getLexema());
            }
        }
    }
}

private Token cadeia() {
```

```
    int estado = 1;
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (estado == 1) {
            if (c == '"') {
                estado = 2;
            } else {
                return null;
            }
        } else if (estado == 2) {
            if (c == '\n') {
                return null;
            }
            if (c == '"') {
                return new Token(TipoToken.Cadeia, ldat.getLexema());
            } else if (c == '\') {
                estado = 3;
            }
        } else if (estado == 3) {
            if (c == '\n') {
                return null;
            } else {
                estado = 2;
            }
        }
    }
}

private void espacosEComentarios() {
    int estado = 1;
    while (true) {
        char c = (char) ldat.lerProximoCaractere();
        if (estado == 1) {
            if (Character.isWhitespace(c) || c == ' ') {
                estado = 2;
            } else if (c == '%') {
                estado = 3;
            } else {
                ldat.retroceder();
                return;
```

```
239                 }
240             } else if (estado == 2) {
241                 if (c == '%') {
242                     estado = 3;
243                 } else if (!(Character.isWhitespace(c) || c == ' ')) {
244                     ldat.retroceder();
245                     return;
246                 }
247             } else if (estado == 3) {
248                 if (c == '\n') {
249                     return;
250                 }
251             }
252         }
253     }
254 
255     private Token palavrasChave() {
256         while (true) {
257             char c = (char) ldat.lerProximoCaractere();
258             if (!Character.isLetter(c)) {
259                 ldat.retroceder();
260                 String lexema = ldat.getLexema();
261                 if (lexema.equals("DECLARACOES")) {
262                     return new Token(TipoToken.PCDeclaracoes, lexema);
263                 } else if (lexema.equals("ALGORITMO")) {
264                     return new Token(TipoToken.PCAlgoritmo, lexema);
265                 } else if (lexema.equals("INTEIRO")) {
266                     return new Token(TipoToken.PCInteiro, lexema);
267                 } else if (lexema.equals("REAL")) {
268                     return new Token(TipoToken.PCReal, lexema);
269                 } else if (lexema.equals("ATRIBUIR")) {
270                     return new Token(TipoToken.PCAtribuir, lexema);
271                 } else if (lexema.equals("A")) {
272                     return new Token(TipoToken.PCA, lexema);
273                 } else if (lexema.equals("LER")) {
274                     return new Token(TipoToken.PCLer, lexema);
275                 } else if (lexema.equals("IMPRIMIR")) {
276                     return new Token(TipoToken.PCImprimir, lexema);
277                 } else if (lexema.equals("SE")) {
```

```
278                     return new Token(TipoToken.PCSe, lexema);
279                 } else if (lexema.equals("SENAO")) {
280                     return new Token(TipoToken.PCSenao, lexema);
281                 } else if (lexema.equals("ENTAO")) {
282                     return new Token(TipoToken.PCEntao, lexema);
283                 } else if (lexema.equals("ENQUANTO")) {
284                     return new Token(TipoToken.PCEnquanto, lexema);
285                 } else if (lexema.equals("INICIO")) {
286                     return new Token(TipoToken.PCInicio, lexema);
287                 } else if (lexema.equals("FIM")) {
288                     return new Token(TipoToken.PCFim, lexema);
289                 } else if (lexema.equals("E")) {
290                     return new Token(TipoToken.OpBoolE, lexema);
291                 } else if (lexema.equals("OU")) {
292                     return new Token(TipoToken.OpBoolOu, lexema);
293                 } else {
294                     return null;
295                 }
296             }
297         }
298     }
299 
300     private Token fim() {
301         int caractereLido = ldat.lerProximoCaractere();
302         if (caractereLido == -1) {
303             return new Token(TipoToken.Fim, "Fim");
304         }
305         return null;
306     }
307 }
```

Código fonte 1 - Gerado por humano

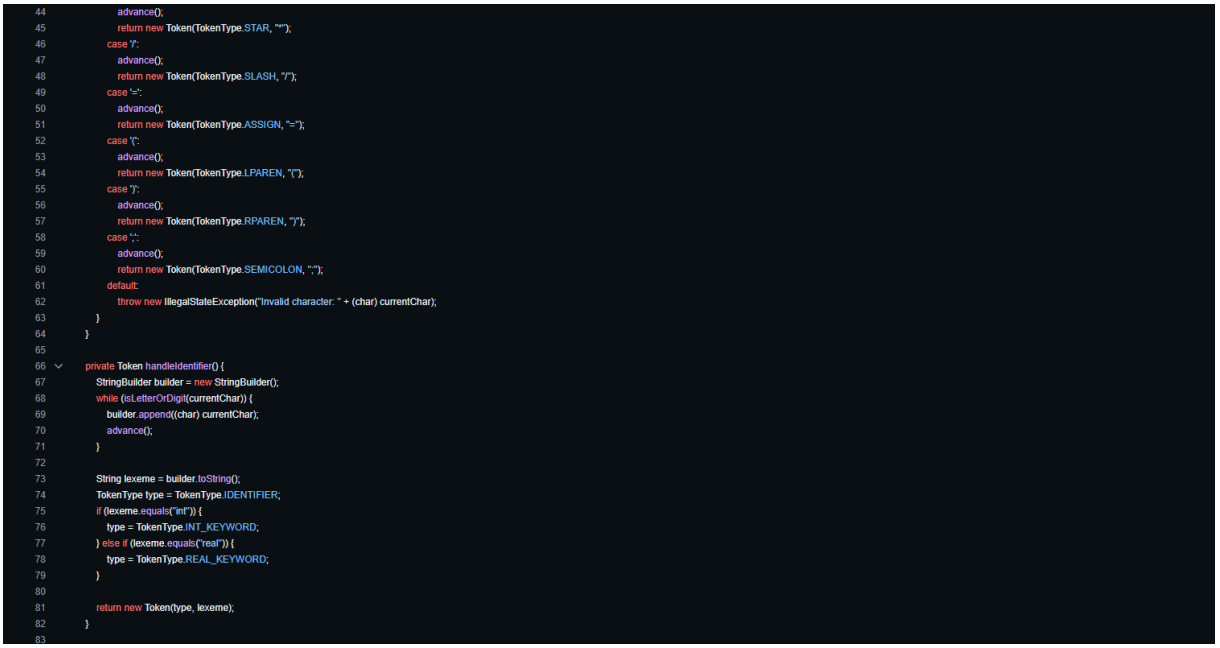
```java
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class LexicalAnalyzer {
    private final List<Token> tokens;
    private String sourceCode;
    private int currentPosition;

    public LexicalAnalyzer(String sourceCode) {
        this.tokens = new ArrayList<>();
        this.sourceCode = sourceCode;
        this.currentPosition = 0;
    }

    public List<Token> analyze() {
        while (currentPosition < sourceCode.length()) {
            char currentChar = sourceCode.charAt(currentPosition);

            if (Character.isDigit(currentChar)) {
                analyzeNumber();
            } else if (Character.isLetter(currentChar)) {
                analyzeWord();
            } else if (currentChar == '"') {
                analyzeString();
            } else if (isOperator(currentChar)) {
                analyzeOperator();
            } else if (isDelimiter(currentChar)) {
                addToken(TokenType.DELIMITER, String.valueOf(currentChar));
                currentPosition++;
            } else if (Character.isWhitespace(currentChar)) {
                currentPosition++;
            } else {
                currentPosition++; // Ignoring unrecognized characters
            }
        }

        return tokens;
    }

    private void analyzeNumber() {
        StringBuilder number = new StringBuilder();
```

```java
        while (currentPosition < sourceCode.length() && Character.isDigit(sourceCode.charAt(currentPosition))) {
            number.append(sourceCode.charAt(currentPosition));
            currentPosition++;
        }
        addToken(TokenType.NUMBER, number.toString());
    }

    private void analyzeWord() {
        StringBuilder word = new StringBuilder();
        while (currentPosition < sourceCode.length() &&
                (Character.isLetterOrDigit(sourceCode.charAt(currentPosition)) || sourceCode.charAt(currentPosition) == '_')) {
            word.append(sourceCode.charAt(currentPosition));
            currentPosition++;
        }
        addToken(TokenType.WORD, word.toString());
    }

    private void analyzeString() {
        currentPosition++; // Move past the initial quote
        StringBuilder string = new StringBuilder();
        while (currentPosition < sourceCode.length() && sourceCode.charAt(currentPosition) != '"') {
            string.append(sourceCode.charAt(currentPosition));
            currentPosition++;
        }
        if (currentPosition < sourceCode.length() && sourceCode.charAt(currentPosition) == '"') {
            addToken(TokenType.STRING, string.toString());
            currentPosition++; // Move past the ending quote
        } else {
            // Unclosed string, do something like throw an error or handle it accordingly
        }
    }

    private void analyzeOperator() {
        char currentChar = sourceCode.charAt(currentPosition);
        StringBuilder operator = new StringBuilder();
        operator.append(currentChar);

        // Check for multi-character operators
        while (currentPosition + 1 < sourceCode.length()) {
```

```
            String possibleOperator = operator.toString() + sourceCode.charAt(currentPosition + 1);
            if (isOperator(possibleOperator)) {
                operator.append(sourceCode.charAt(currentPosition + 1));
                currentPosition++;
            } else {
                break;
            }
        }
        addToken(TokenType.OPERATOR, operator.toString());
        currentPosition++;
    }

    private boolean isOperator(char c) {
        String[] operators = {"+", "-", "*", "/", "=", "<", ">", "!", "&", "|"};
        return new String(operators).contains(String.valueOf(c));
    }

    private boolean isOperator(String str) {
        String[] operators = {"==", "!=", "<=", ">=", "&&", "||"};
        return new String(operators).contains(str);
    }

    private boolean isDelimiter(char c) {
        return c == ';' || c == ',' || c == '(' || c == ')' || c == '{' || c == '}';
    }

    private void addToken(TokenType type, String value) {
        tokens.add(new Token(type, value));
    }
}
```

Código fonte 2 - Gerado pelo ChatGPT

```
import java.io.IOException;

import Reader;

public class LexicalAnalyzer {

    private Reader reader;
    private int currentChar;

    public LexicalAnalyzer(Reader reader) {
        this.reader = reader;
        try {
            currentChar = reader.read();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Token nextToken() throws IOException {
        while (isWhitespace(currentChar)) {
            advance();
        }

        if (currentChar == -1) {
            return new Token(TokenType.EOF, null);
        }

        if (isLetter(currentChar)) {
            return handleIdentifier();
        }

        if (isDigit(currentChar)) {
            return handleNumber();
        }

        switch (currentChar) {
            case '+':
                advance();
                return new Token(TokenType.PLUS, "+");
            case '-':
                advance();
                return new Token(TokenType.MINUS, "-");
            case '*':
```



```
            advance();
            return new Token(TokenType.STAR, "*");
        case '/':
            advance();
            return new Token(TokenType.SLASH, "/");
        case '=':
            advance();
            return new Token(TokenType.ASSIGN, "=");
        case '(':
            advance();
            return new Token(TokenType.LPAREN, "(");
        case ')':
            advance();
            return new Token(TokenType.RPAREN, ")");
        case ';':
            advance();
            return new Token(TokenType.SEMICOLON, ";");
        default:
            throw new IllegalStateException("Invalid character: " + (char) currentChar);
    }
}

private Token handleIdentifier() {
    StringBuilder builder = new StringBuilder();
    while (isLetterOrDigit(currentChar)) {
        builder.append((char) currentChar);
        advance();
    }

    String lexeme = builder.toString();
    TokenType type = TokenType.IDENTIFIER;
    if (lexeme.equals("int")) {
        type = TokenType.INT_KEYWORD;
    } else if (lexeme.equals("real")) {
        type = TokenType.REAL_KEYWORD;
    }

    return new Token(type, lexeme);
}
```

```
private Token handleNumber() {
    StringBuilder builder = new StringBuilder();
    while (isDigit(currentChar)) {
        builder.append((char) currentChar);
        advance();
    }

    if (currentChar == '.') {
        builder.append('.');
        advance();

        while (isDigit(currentChar)) {
            builder.append((char) currentChar);
            advance();
        }

        return new Token(TokenType.REAL_NUMBER, builder.toString());
    } else {
        return new Token(TokenType.INT_NUMBER, builder.toString());
    }
}

private void advance() throws IOException {
    currentChar = reader.read();
}

private boolean isWhitespace(int c) {
    return c == ' ' || c == '\t' || c == '\n' || c == '\r';
}

private boolean isLetter(int c) {
    return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');
}

private boolean isDigit(int c) {
    return c >= '0' && c <= '9';
}

private boolean isLetterOrDigit(int c) {
    return isLetter(c) || isDigit(c);
```

```
    }
}
```

Código fonte 3 - Gerado pelo Bard